UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Miloš Jovanov

Primerjava orodij za razvoj mobilnih aplikacij

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

Ljubljana, 2017

UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Miloš Jovanov

Primerjava orodij za razvoj mobilnih aplikacij

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: doc. dr. Aljaž Zrnec

Ljubljana, 2017

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo: Primerjava orodij za razvoj mobilnih aplikacij

Tematika naloge:

Sodobne mobilne naprave so pomemben del vsakdanjika skoraj vsakega posameznika. Naloga razvijalcev je poenotiti uporabnost aplikacij na različnih mobilnih platformah in jim ponuditi enako izkušnjo. Razvijalci morajo tako biti vešči več programskih jezikov in znati uporabljati različna okolja za razvoj. Xamarin studio je močno razvojno okolje, ki omogoča razvoj aplikacij za več platform hkrati. To pomeni, da v okviru enega projekta lahko nastane koda za različne mobilne operacijske sisteme. V okviru diplomskega dela prikažite razvoj domorodne aplikacije za Android in iOS. Nato v okolju Xamarin razvijte enako aplikacijo, vendar z deljeno kodo. Predstavite prednosti in slabosti posamičnega razvojnega okolja ter procesa razvoja aplikacije z različnimi orodji.

Kazalo

Kazalo slik

Seznam uporabljenih kratic

kratica	angleško	slovensko
MVC	Model-view-controller	Model-pogled-kontroler
SQL	Structured Query Language	Strukturirani povpraševalni jezik
XML	Extensible Markup Language	Razširljiv označevalni jezik
XAML	Extensible Application Markup Language	Razširljiv aplikacijski označevalni jezik

Povzetek

Naslov: Primerjava orodij za razvoj mobilnih aplikacij

Namen diplomske naloge je primerjati razvojna orodja Android studio, Xcode in Xamarin studio. Vsa tri okolja omogočajo razvoj mobilnih aplikacij. Na trgu in po priljubljenosti med uporabniki prevladujejo mobilne naprave, ki uporabljajo operacijski sistem Android in iOS. Cilj razvijalcev aplikacij je podpreti oba vodilna operacijska sistema in omogočiti enako uporabnost vsem uporabnikom mobilnih naprav. V Android studiu smo v praktičnem delu diplomske naloge razvili aplikacijo za operacijski sistem Android. Okolje Xcode je namenjeno razvoju iOS aplikacije, ki je prav tako prikazan v praktičnem delu. Obe okolji sta namenjeni razvoju domorodnih aplikacij v enem od operacijskih sistemov. To pomeni, da kode med njima ne moremo prenašati in aplikacijo je potrebno vedno znova razviti v obeh okoljih.

Z uporabo orodja Xamarin studio se problema ločenega razvoja lahko znebimo. Omenjeno okolje namreč omogoča razvoj aplikacije za oba operacijska sistema hkrati in na ta način omogoči hitrejše izvajanje procesa razvoja aplikacije. V praktičnem delu diplomskega dela so tako prikazane tudi prednosti in slabosti Xamarin studia v primerjavi z domorodnima orodjema Android studio in Xcode.

Naj opozorimo, da Android studio ni edino orodje za razvoj Android aplikacij. Programski jezik, ki se uporablja v okviru razvoja, je Java. Za sestavljanje grafičnega vmesnika se uporablja jezik XML. Grafični vmesnik je ustvarjen z uporabo grafičnega urejevalnika izgleda (metoda povleci in spusti) ter ročnega pisanja kode v XML. V nasprotju z Android studiem je edino domorodno okolje za razvoj aplikacij iOS orodje Xcode. Ta nam omogoča izbiro med dvema programskima jezikoma, in sicer Objective-C in Swift. V diplomski nalogi smo uporabili programski jezik Swift. Za grafični vmesnik smo uporabili grafični urejevalnik izgleda z metodo povleci in spusti. V ozadju tudi Xcode uporablja XML. Xamarin studio za razvoj uporablja programski jezik C#. Grafični vmesnik se tu sestavlja v celoti v urejevalniku XAML in ustvarjanje grafičnega vmesnika s pristopom povleci in spusti kot pri drugih dveh ni mogoče.

Ključne besede: Android studio, Xamarin, Xcode, Swift, Java, C#, XAML, XML, Android, iOS

Abstract

Title: Comparison of tools for mobile application development

The aim of this thesis is the comparison of development tools between Android Studio, Xcode and Xamarin Studio. All three environments enable the development of mobile applications. Mobile devices that use the operating systems Android and iOS dominate the market and popularity among users. The goal of applications is to support both leading operating systems and allow equal functionality to all users of mobile devices. In practical part of the thesis we will develop an application for the Android operating system using Android Studio. Xcode environment is intended for developing iOS applications, which will also be presented in the practical work. Both of these environments are specifically created for the development of applications in one of the operating systems. This means that the code cannot be transmitted between the environments and it is necessary to develop each application again in both environments.

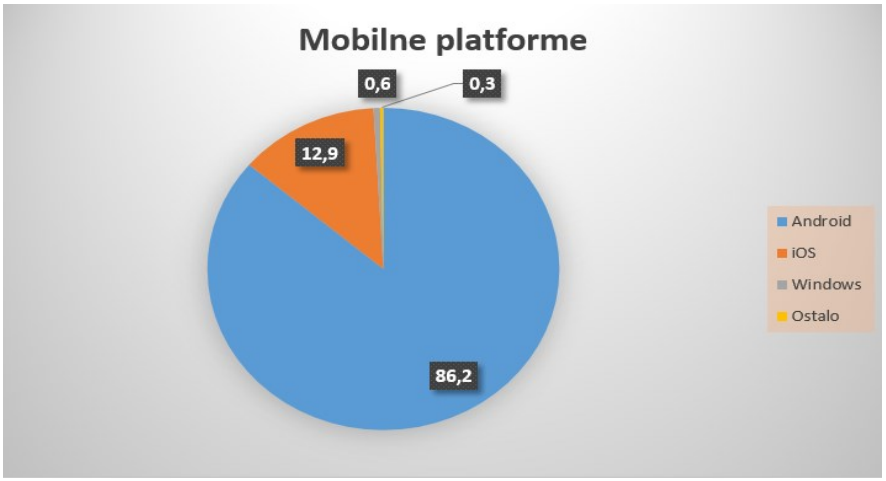
However by using Xamarin Studio this problem is solved. Xamarin enables the development of applications for both operating systems at the same time so the developer speeds up development itself and eliminates duplication of work. In the practical part we will show the advantages and disadvantages of Xamarin Studio compared to Android Studio and Xcodom that target each of its operating systems.

It should be noted that Android Studio is not the only choice for the development of Android applications. The programming language is Java. For compiling the graphical interface we use XML. The graphical interface was created with drag and drop method and with writing XML. In contrast to Android Studio, the only development environment for iOS applications in addition to Xamarin Studio is Xcode. This allows us to choose between two programming languages. These are the Objective-C and Swift. In this thesis, we used Swift. For graphical interface, we used the drag-and-drop method. In the background, Xcode uses XML as well. For the development of the programming, Xamarin Studio uses language C #. Here the graphical interface consists entirely in XAML editor and creating a graphical interface with drag and drop method isn't enabled, as with the other two.

Keywords: Android studio, Xamarin, Xcode, Swift, Java, C#, XAML, XML, Andorid, iOS

Poglavje 1 **Uvod**

V zadnjih letih so mobilne naprave postale najbolj uporabljene naprava za dostopanje do spleta. Razlog je seveda velik napredek tehnologije na področju mobilnih naprav ter njihova enostavna in prijazna uporaba. Na trgu se posledično odvijajo hudi konkurenčni boji med proizvajalci mobilnih naprav. Na osebnih računalnik poznamo več različnih operacijskih sistemov, v mobilnem svetu pa sta vodilna operacijska sistema Android in iOS. V primerjavi z ostalimi operacijskimi sistemi je Android med uporabniki daleč najbolj razširjen, sledi pa mu iOS. Deleže mobilnih naprav z določenim operacijskim sistemom prikazuje Slika 1.1. Kljub temu da na mobilnih napravah zaenkrat prevladuje Android, uporabnikov iOS nikakor ne smemo zanemariti. Ti namreč v povprečju porabijo več denarja za plačljive aplikacije in dodatke, s čemer nadoknadijo visoko prevlado Androida.



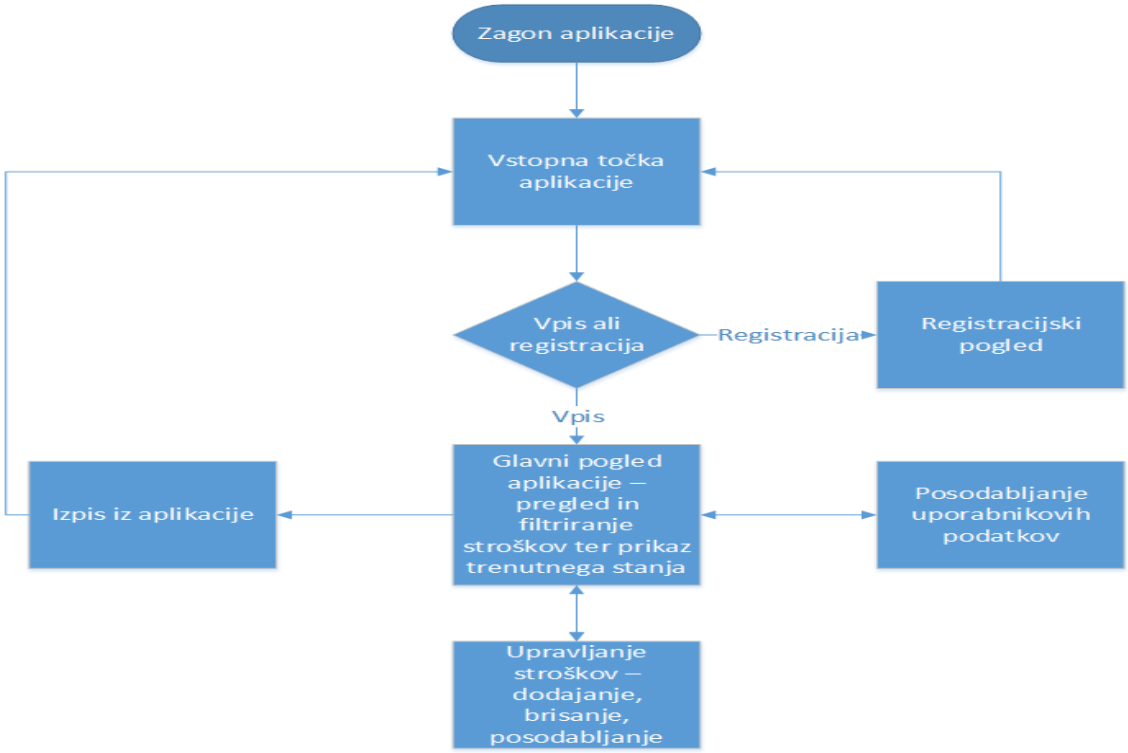
Slika 1.1: Deleži mobilnih naprav z določenim operacijskim sistmemom.

V okviru diplomske naloge se bomo postavili v vlogo razvijalca, ki mora razviti aplikacijo za obe platformi – Android in iOS. Aplikacijo za Android bo potrebno razviti v okolju Android studio in programskem jeziku Java. Razvoj aplikacije za iOS bo potekal v okolju Xcode in programskem jeziku Swift. Po končanem razvoju obeh aplikacij bo sledil še razvoj v okolju Xamarin studio. Ta uporablja programski jezik C#. Prednost Xamarin studia je v večplatformskem razvoju hkrati. V sklopu enega projekta in ene kode bosta sočasno nastali dve aplikaciji za obe zgoraj omenjeni platformi. Videz in funkcionalnosti bomo poizkusili poenotiti

na obeh platformah in na vseh projektih. Zamišljena aplikacija se imenuje Cost Note (glej poglavje 1.1). Gre za aplikacijo za beleženje stroškov, ki lahko nastanejo pri posamezniku na delovnem mestu. Aplikacija bo podatke shranjevala lokalno in uporabljala podatkovno bazo SQLite [20]. Uporabnik aplikacije bo imel dostop do kamere, s katero bo simulirano fotografiranje računa. Aplikacija bo skrbela za pregled nad nastalimi stroški.

1.1 Funkcionalne specifikacije aplikacije

Namen aplikacije je izdelati poslovno aplikacijo za beleženje stroškov. Velika večina podjetji stroške beleži na papirnih obrazcih, ki jih zaposleni nato oddajajo v nadaljnjo obdelavo. S to aplikacijo želimo poenostaviti ta zamudni korak. Zaposlenemu bomo omogočili vnašanje vseh potrebnih podatkov preko aplikacije. Podatki se bodo hranili lokalno na napravi uporabnika. V primeru razvoja prave aplikacije bi seveda podatke hranili na strežniku določenega podjetja. Aplikacija bo zahtevala registracijo uporabnika. Uporabnik bo znotraj aplikacije lahko svoje podatke posodabljal. Glavno okno aplikacije bo služilo za pregled stroškov. Vsebovalo bo seznam stroškov in prikaz trenutnega stanje aktivnih in povrnjenih stroškov. Uporabnik bo lahko seznam stroškov filtriral po mesecih. Ob dodajanju novega stroška bo uporabnik z uporabo kamere slikal račun stroška. Funkcionalne zahteve aplikacije so prikazane na spodnji sliki (Slika 1.2).



Slika 1.2: Prikaz uporabe aplikacije na diagramu.

Poglavje 2 **Uporabljene naprave in tehnologije**

Razvoj je potekal na osebnem računalniku z operacijskim sistemom OSX, ki omogoča razvoj aplikacije iOS in z uporabo Xamarin studia. Vsa tri razvojna okolja Android studio, Xcode in Xamarin studio so prav tako podprta na operacijskem sistemu Windows. Xamarin studio je na operacijskem sistemu Windows podprt v orodju Microsoft Visual Studiu in ne obstaja kot ločeno razvojno okolje. Če želimo pri razvoju iOS aplikacij dostopati do kamer, potrebujemo fizično mobilno napravo proizvajalca Apple. Ta naprava je lahko Iphone ali Ipad. Brez fizične mobilne naprave lahko uporabimo iOS emulator. Za delujoč iOS emulator potrebujemo osebni računalnik, ki uporablja operacisjki sistem OSX. Mobilno aplikacijo iOS je možno razviti tudi na operacijskem sistemu Windows z pomočjo razvojnega okolja Visual studio. Dostopno mora razvijalec imeti osebni računalnik z operacijskim sistemom OSX, kjer deluje iOS emulator. V Visual studiu (operacijski sitem Windows) se povežemo preko spletne povezave na osebni računalnik z operacijskim sistemom OSX (Mac), kjer razvojno okolje zazna emulator iOS kot svoj lastni. Android emulator deluje na operacijskem sistemu OSX kot operacijskem sistemu Windows. Za dostop do kamere Android emulaotor uporabi kamero osebnega računalnika na katerem deluje, če je ta na voljo. Za razvoj komercialnih aplikacij se vedno uporabljajo fizične naprave za končna testiranja.

2.1 Emulatorji in naprave

V sklopu diplomske naloge smo uporabili fizično napravo Apple Iphone 6s (iOS). To pomeni, da smo aplikacijo namestili na fizično napravo in jo na njej tudi testirali. V začetnih fazah razvoja aplikacije smo uporabili tudi emulator za iOS. Android aplikacijo smo v celoti testirali v emulatorju.

2.1.1 Kaj je emulator?

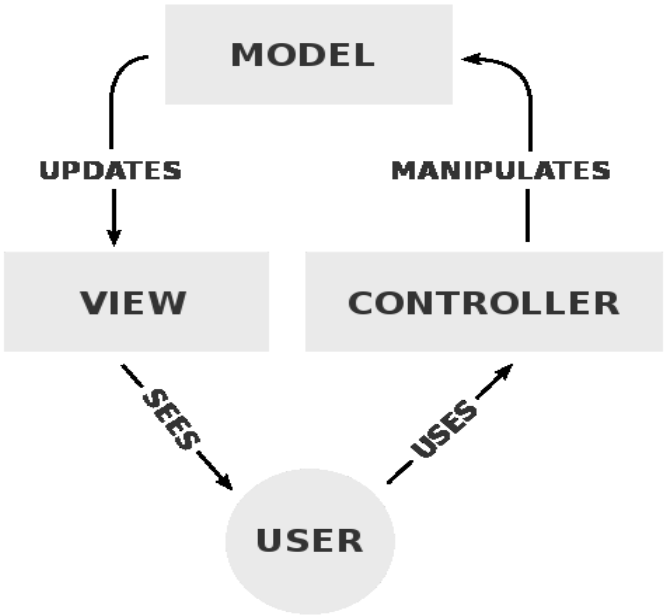
Emulator [17] je nadomestek fizične mobilne naprave, v katerem s tipkovnico in kliki miške simuliramo dotik osebe. Emulatorji se izvajajo na osebnem računalniku razvijalca in omogočajo uporabo specifične različice nekega mobilnega operacijskega sistema brez fizične mobilne naprave. Lahko imajo različne velikosti zaslona kakor prave fizične mobilne naprave. Še posebej so primerni za testiranje grafičnega vmesnika. Z malo truda lahko testiramo videz aplikacij na različnih mobilnih napravah z različnimi različicami operacijskih sistemov. Nekateri imajo dostop tudi do vhodnih naprav računalnika, na katerih se izvajajo (kamera, mikrofon in disk).

2.2 Model-pogled-krmilnik (MVC)

Model-pogled-krmilnik (angleško Model-view-controller) ali skrajšano MVC [19] je arhitekturni vzorec za razvoj programske opreme. Pristop MVC je globalno zelo pogost pristop razvoja spletnih in mobilnih aplikacij. Njegove glavne komponente so:

- **Model**: Predstavlja najnižji nivo in je odgovoren za ravnanje s podatki.
- **Pogled**: Nivo pogleda je odgovoren za prikaz podatkov uporabniku.
- **Krmilnik**: Ta nivo vsebuje logični del aplikacije. Skrbi za komunikacijo in interakcijo med nivojem Modela in Pogleda. Nadzoruje izvajanje aplikacije.

Pristop MVC loči logični del aplikacije od grafičnega in omogoča neodvisen razvoj. Krmilnik upravlja z vsemi zahtevami aplikacije. Glede na zahtevo uporabnika logični del aplikacije pripravi model z ustreznimi podatki. Tega nato pošlje v pogled, kjer se podatki ustrezno prikažejo uporabniku. Postopek je viden na sliki (Slika 2.1).



Slika 2.1: Prikaz delovanja arhitekture MVC.

2.3 Programski jeziki

Razvoj aplikacij je potekal v treh različnih programskih jezikih [22], kateri so vsi visokonivojski. Visokonivojski programski jeziki so za človeka bolj berljivi od nizkonivojskega

in preprostejši za uporabo. Potrebujejo prevajalca, ki je del razvojnih okolij. V sklopu diplomske naloge sem uporabil sledeče programske jezike:

- **Java**
- **Swift**
- **C#**

2.3.1 Java

Razvoj Android aplikacije je potekal v programskem jeziku Java [14], ki je objektno usmerjen programski jezik. Namen Jave je enkratno prevajanje kode, ki je ni potrebno ponovno prevajati na različnih sistemih. Prevedena aplikacija lahko deluje na vseh napravah, ki podpirajo Javo. Ta programski jezik je bil razvit na osnovi programskega jezika C.

2.3.2 Swift

Programski jezik Swift [3] sem uporabil za razvoj aplikacije iOS v okolju Xcode. Swift je razvit na osnovi programskega jezika C. Je med mlajšimi programskimi jeziki. Aplikacije, razvite v Swiftu, lahko delujejo na vseh napravah proizvajalca Apple Inc. Je prijazen za razvijalce, ki še nimajo veliko izkušenj. Swift omogoča projekte tipa igrišče (angleško palyground), pri katerih se koda izvaja skriptno in prevajanje kode ni potrebno. Rezultati so vidni v realnem času. Prednost Swifta je preprosta sintaksa v primerjavi z programskim jezikom Objective-C, ki je alternativa razvoju aplikacij v iOS operacijskem sistemu. Swift je prav tako objektno usmerjen programski jezik. Trenutna različica je Swift 3.

2.3.3 C#

Deljena aplikacija diplomske naloge je bila s pomočjo okolja Xamarin studio razvita v programskem jeziku C#, ki je prav tako objektno usmerjen programski jezik. Tako kot predhodno omenjena programska jezika je razvit na osnovi programskega jezika C. C# uporablja Microsoftovo ogrodje .NET [1]. Trenutna različica jezika je 6.0.

2.4 XML

XML je razširljiv označevalni jezik s fleksibilno in preprosto strukturo . Označevalni jezik se uporablja za opis in predstavitev podatkov. Velik delež strukturiranih podatkov na spletu se nahaja v obliki XML. V diplomski nalogi smo s pomočjo XML razvili grafični vmesnik

aplikacije Android. Osnova dokumenta XML so njegovi elementi, ki imajo lahko poljubna imena. Potrebno je pravilo, da vedno pravilno končamo/zapremo element. Elemente lahko gnezdimo poljubno. Strukturo preprostega XML lahko vidimo na sliki (Slika 2.2). Ta primer prikazuje XML, ki bi ga lahko uporabili za pošiljanje podatkov. Poslane podatke bi aplikacija lahko prevedla v svoj lokalni format. To je ena izmed glavnih prednosti standarda XML, saj poenoti strukturo podatkov.

```
<?xml version="1.0"?>
<Oseba>
    <Ime>John Doe</Ime>
    <Spol>Moški</Spol>
    <Starost>23</Starost>
</Oseba>
```

Slika 2.2: Primer podatkovne strukture XML.

V prikazanem primeru (Slika 2.3) vidimo uporabo standarda XML za prikaz grafičnega vmesnika v Android studiu. V njem so elementi opredeljeni vnaprej in predstavljajo gradnike aplikacije. Tem elementom lahko nastavimo določene lastnosti ali atribute, s katerimi vplivamo na njegov izgled. Za primer lahko vzamemo element tekstovnega polja. Preko lastnosti lahko definiramo barvo besedila, njegove odmike in velikost.

```
<TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginLeft="14dp"
        android:layout_marginTop="18dp"
        android:text="@string/neko besedilo" />
```

Slika 2.3: Primer uporabe standarda XML za prikaz gradnika v grafičnem vmesniku Android aplikacije.

2.5 XAML

Za razvoj grafičnega vmesnika skupne aplikacije v okolju Xamarin studio smo uporabili označevalni jezik XAML [18], ki je bil razvit na osnovi XML. Tega je Microsoft razvil kmalu po razvoju programskega jezika C#. Integriran je v številna Microsoftova ogrodja kot so:

- .NET,
- WPF,
- SilverLight,
- UWP,
- Windows Phone in
- Xamarin.Forms.

Med zgoraj naštetimi ogrodji obstajajo razlike. Te najhitreje opazimo pri imenih in lastnostih gradnikov, ki se preslikajo v objekte programskega jezika. Vsako ogrodje podpira svoj XAML, ki je v podanih okoliščinah specifične platforme maksimalno poenoten. Namen XAML je ločevanje grafičnega vmesnika in aplikacijske logike. Grafični vmesnik je mogoče razviti tudi s pomočjo programskega jezika, kar pa je bolj kompleksno in od razvijalca zahteva več izkušenj. Na sliki (Slika 2.4) lahko vidimo primer vnosnega polja, opredeljenega v formatu XAML.

```
<TextBox Text="Testno besedilo"
         TextAlignment="Center"
         Height="20"
         Width="auto"
         VerticalAlignment="Top"/>
```

Slika 2.4: Primer vnosnega polja z vsebino »Testno besedilo«.

2.6 SQLite

SQLlite [20] je knjižnica, ki predstavlja relacijsko podatkovno bazo. Njena posebnost je, da ne potrebuje strežnika (SUPB), s katerega je dostopna aplikaciji. Ni je potrebno konfigurirati. Nameščena je znotraj aplikacije in omogoča lokalno shranjevanje podatkov. Informacije se hranijo na pomnilniku naprave. SQLite je odprtokodna knjižnica in je na voljo tako za poslovne kot tudi zasebne namene. Hkrati je najbolj pogosto uporabljena podatkovna baza na mobilnih napravah. Dostopna je na različnih mobilnih platformah, zaradi česar sem jo uporabil v diplomski nalogi. Implementacije SQLite se na vsaki mobilni platformi nekoliko razlikujejo. Delovanje baze je v ozadju enako. Slabost SQLite baze je, da v primeru izbrisa aplikacije izgubimo vse podatke. Ob ponovni namestitvi se namesti nova instanca baze, ki obstaja do izbrisa aplikacije.

Poglavje 3 **Razvojna okolja**

V poglavju bomo predstavili vsa tri razvoja okolja (angleško integrated development enviroment - IDE), v katerih smo razvili aplikacijo. Razvojno okolje [12] je programska oprema, ki na enem mestu združuje vsa potrebna orodja za razvoj aplikacij. V razvojno okolje Visual Studio lahko platformo Xamarin vključimo kot dodatek. Na operacijskem sistemu OSX je platforma Xamarin samostojna aplikacija (Xamarin studio). Prijazen izgled grafičnega vmesnika lahko naredi posamezno okolje bolj priljubljeno med uporabniki. Ključna orodja, ki jih vsebuje vsako razvojno okolje so:

- Prevajalnik (angleško Compiler),
- Urejevalnik kode,
- Razhroščevalnik (angleško Debugger) in
- Grafični vmesnik za upravljanje okolja (angleško Graphical User Interface).

V tem poglavju bom predstavil grafični vmesnik posameznega okolja in njegove funkcionalnosti ter poudaril posebnosti. Prav tako bom opisal postopek kreiranja novega projekta v vsakem okolju.

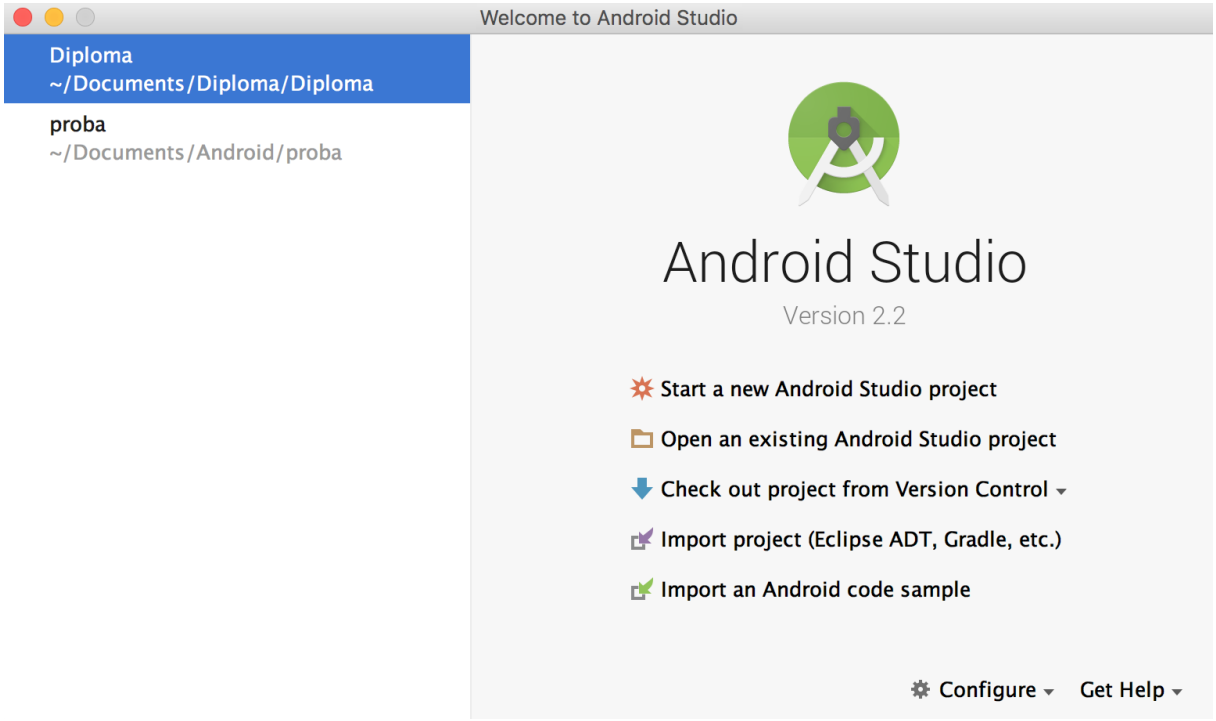
3.1 Android studio

Android studio [6] je uradno razvojno okolje za razvoj Android aplikacij. Zasnovan je na urejevalniku kode InteliJ IDEA [15]. Namestimo ga lahko na osebne računalnike z operacijskim sistemom Microsoft Windows, OSX (Mac) in Linux. To razvojno okolje za razvijanje zalednega dela aplikacije (logike) uporablja programski jezik Java. Grafični vmesnik aplikacije se razvija v standardu XML. Razvijalec lahko videz aplikacije zgradi preko grafičnega vmesnika znotraj razvojnega okolja (angleško drag & drop) ali s pisanjem v datoteko XML. Razvijalci običajno uporabljajo kombinacijo obeh načinov. Google je lastnik Android studia in posledično ima to razvojno okolje zelo dobro podprte knjižnice za uporabo Googlovih storitev. Vse Android emulatorje, ki so na voljo zntoraj razvojnega okolja Android studio je prav tako razvil Google. Danes obstaja 25 različic operacijskega sistema Android. Android studio uporablja Android SDK (angleško software development kit), v okviru katerega določimo različico izbranega sistema Android. Znotraj SDK so priložene vse potrebne knjižnice, podprte teme, razhroščevalnik in primeren emulator. Trenutno je na trgu različica Android SDK 25.2.3

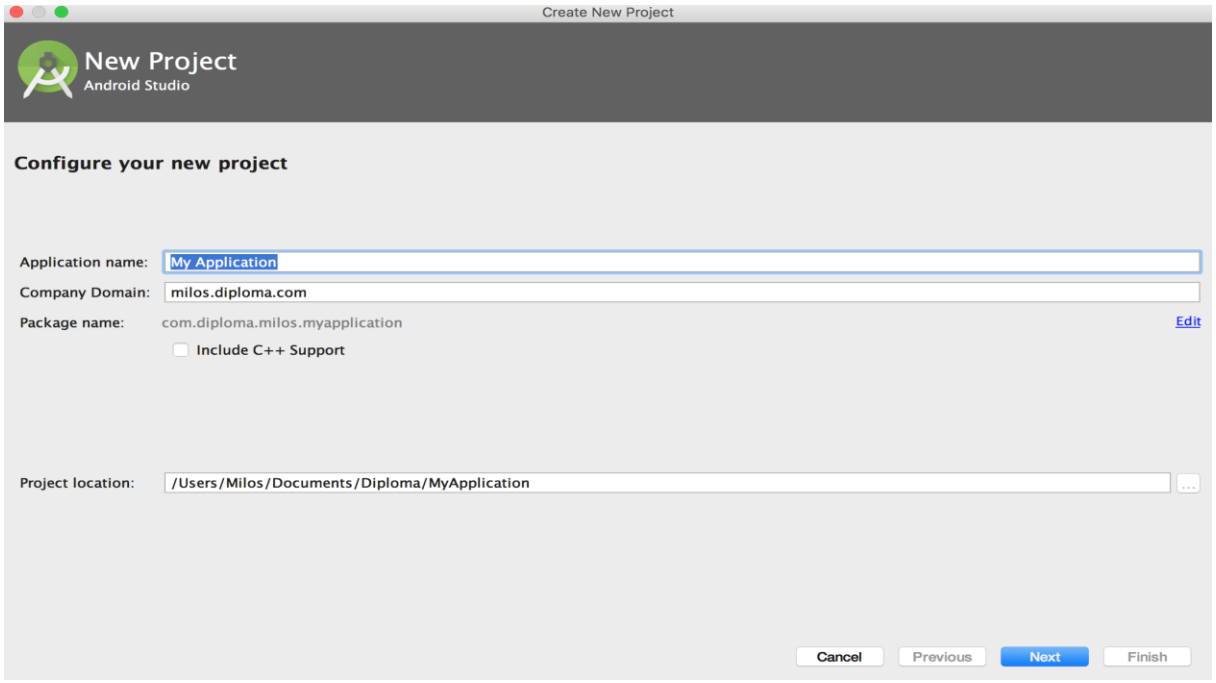
[5]. S hitrim razvojem tehnologije se možnosti in funkcionalnosti orodja širijo. Vse naprave ne podpirajo vseh različic. Razvijalec mora vedno določiti ciljno različico razvoja in minimalno podrto različico. Zaradi nekompatibilnosti naprav, moramo biti pri razvoju pazljivi in ustrezno določiti različice. V diplomski nalogi je ciljna različica aplikacije 23 in minimalno podprta različica 15.

3.1.1 Kreiranje novega projekta

Na spodnji sliki (Slika 3.1) je viden začetni pogled razvojnega okolja. V pogledu izberemo že obstoječ projekt ali ustvarimo novega. Razvojno okolje nas vodi skozi oba koraka. Ob izbiri novega projekta se odpre okno (Slika 3.2), v katerem izberemo ime projekta in njegovo lokacijo na disku računalnika.

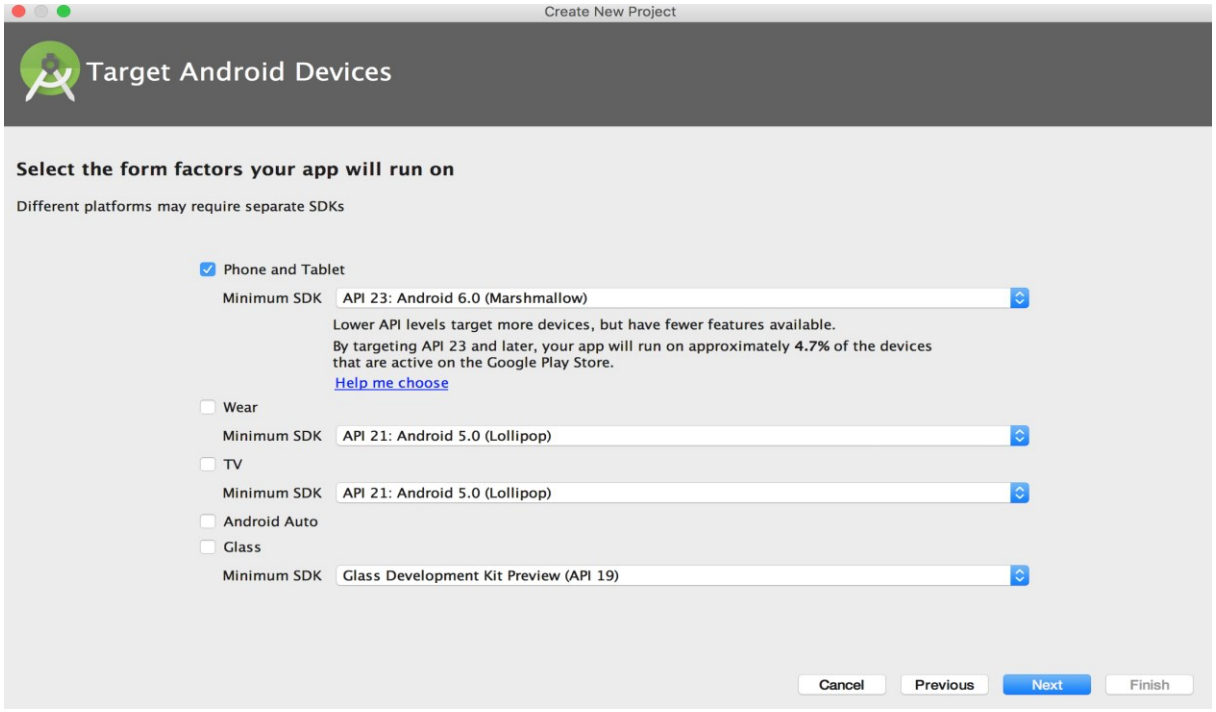


Slika 3.1: Začetni pogled okolja Android studio.



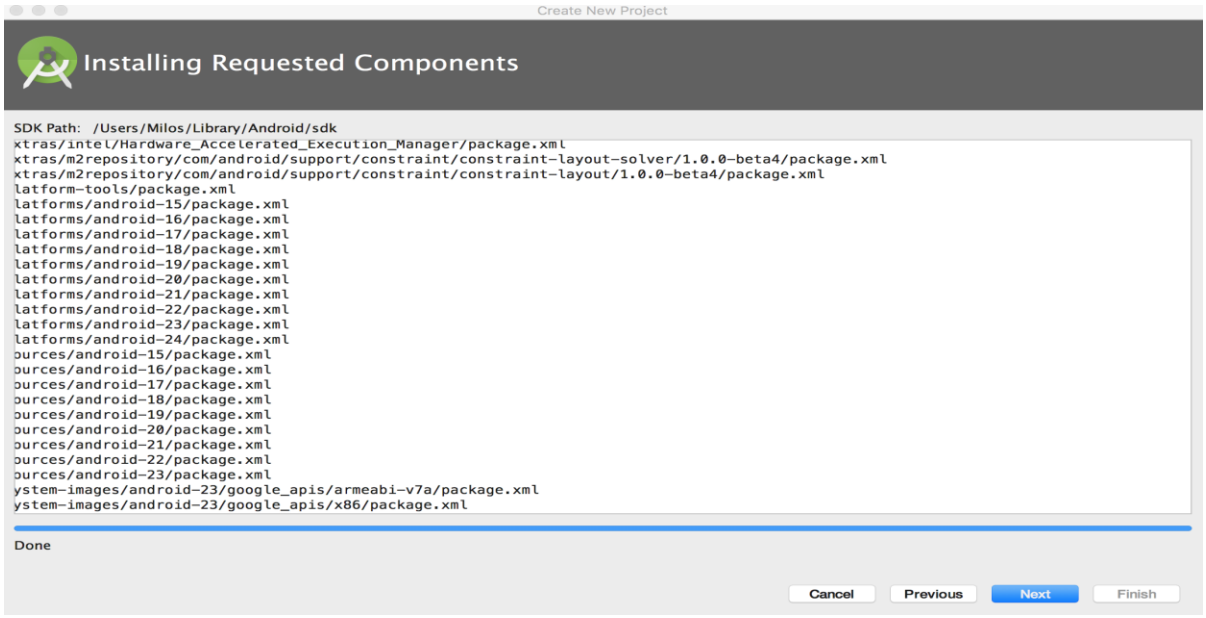
Slika 3.2: Poimenovanje novega projekta v Android studiu in izbira mesta, na katerem bo projekt shranjen na disku računalnika.

Pri naslednjem koraku (Slika 3.3) moramo biti pazljivi, saj tu določimo ciljni SDK aplikacije. Prav tako na tem mestu določimo minimalno podprt SDK.



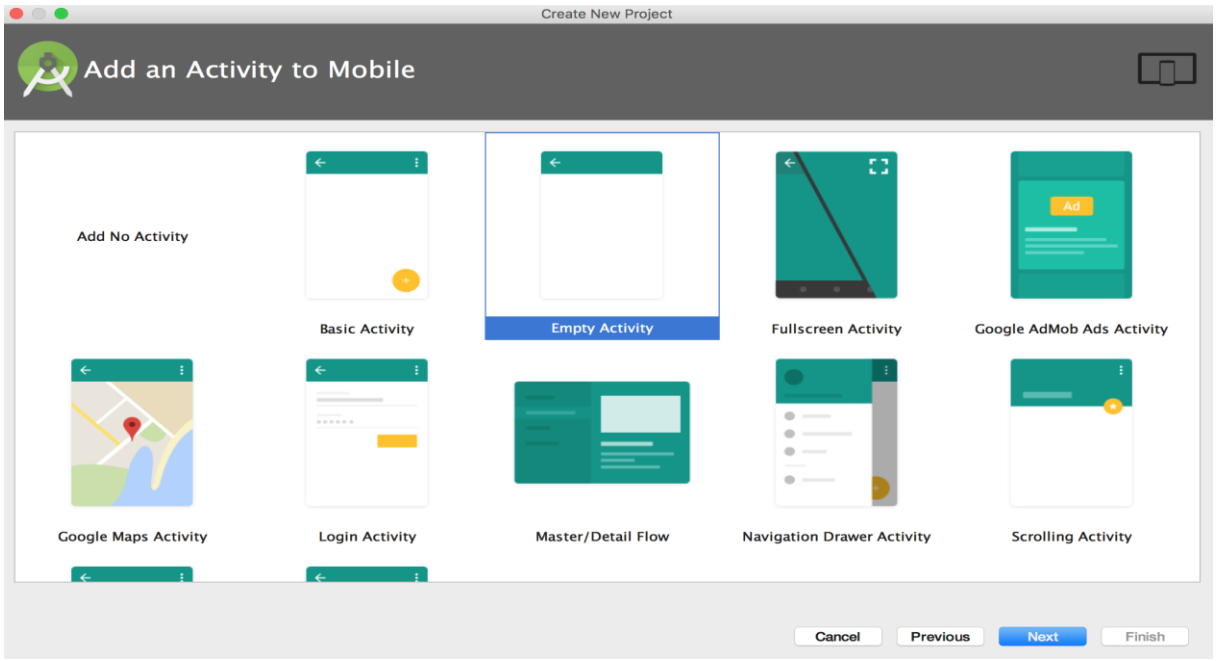
Slika 3.3: Izbira ciljnega SDK.

Po izbiri SDK-ja okolje s spleta prenese vse potrebne knjižnice in datoteke (Slika 3.4). Ta korak se zgodi ob prvem kreiranju aplikacije poljubnega SDK. Ob kreiranju druge aplikacije enakega ciljnega SDK okolje knjižnic ne bo ponovno prenašalo.



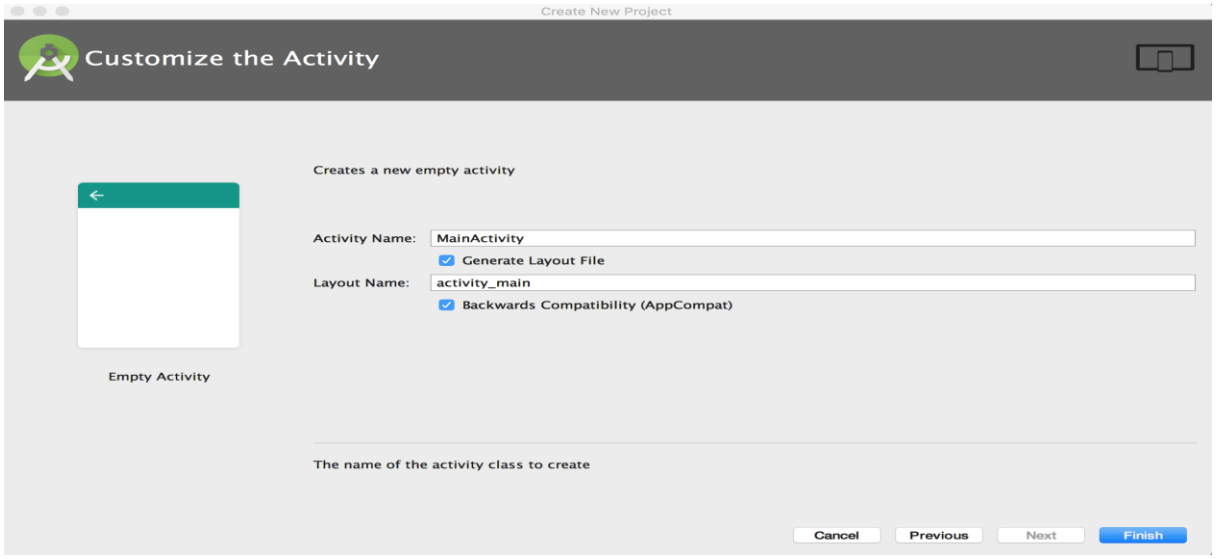
Slika 3.4: Primer prenašanja vseh potrebnih knjižnic in datotek.

Sledi izbira predloge aplikacije. Okolje nam ponudi nekaj predpripravljenih predlog (Slika 3.5).



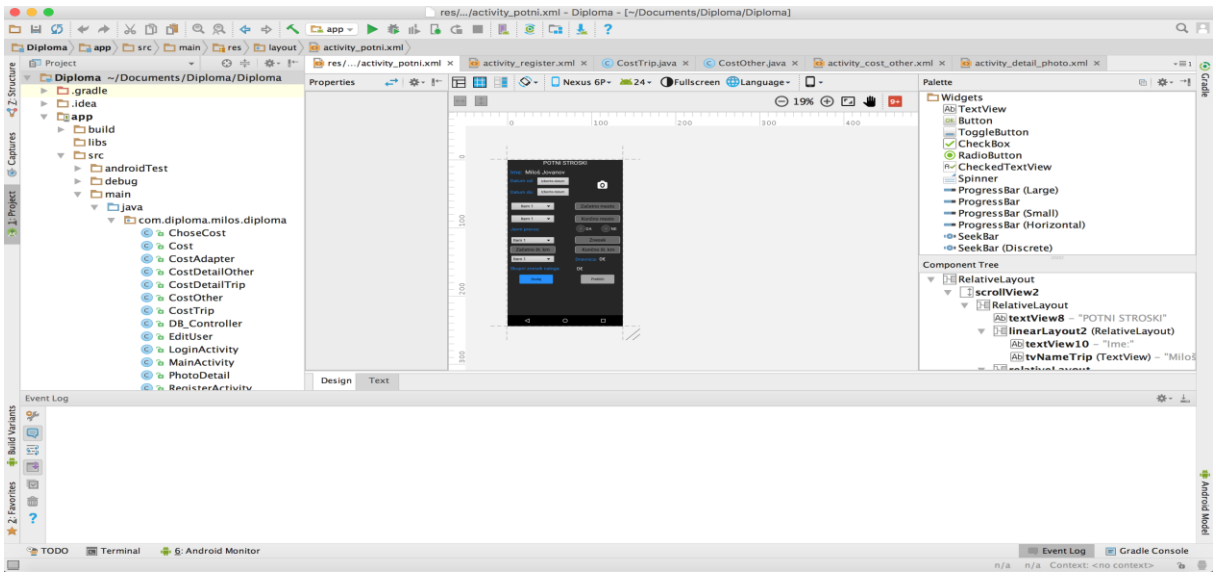
Slika 3.5: Izbira predloge aplikacije.

Razvijalci navadno izberejo prazen pogled (angleško Empty Activity). V aplikaciji je sinonim za pogled v okolje Android studio Activity (slovensko dejavnost). Sledi poimenovanje tega pogleda (Slika 3.6). Okolje bo ustvarilo dve datoteki. Prva je za grafični vmesnik tega pogleda in je tipa XML. Druga skrbi za logično delovanje tega pogleda in je javanska datoteka. Vsak pogled mora vsebovati obe datoteki.



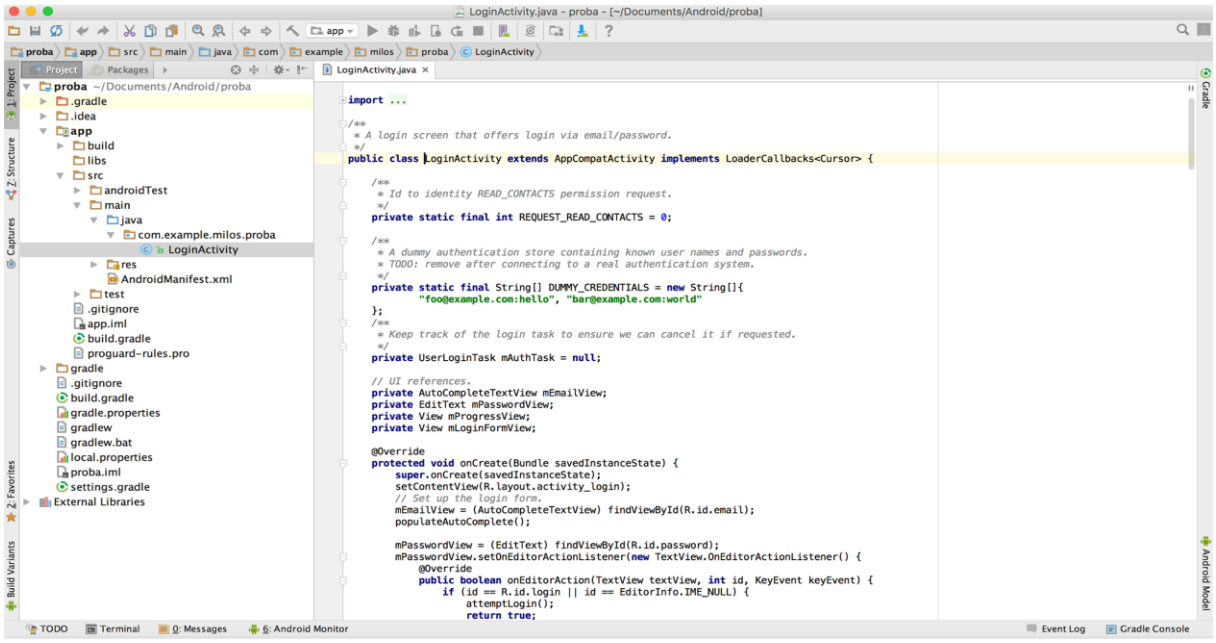
Slika 3.6: Poimenovanje pogleda (Activity).

Po tem koraku se odpre osrednje okno razvojnega okolja Android studio (Slika 3.7). V primeru odprtja že obstoječega projekta vse zgoraj opisane korake preskočimo. Okolje nas nato preusmeri na osrednje okno.



Slika 3.7: Osrednje okno razvojnega okolja Android studia.

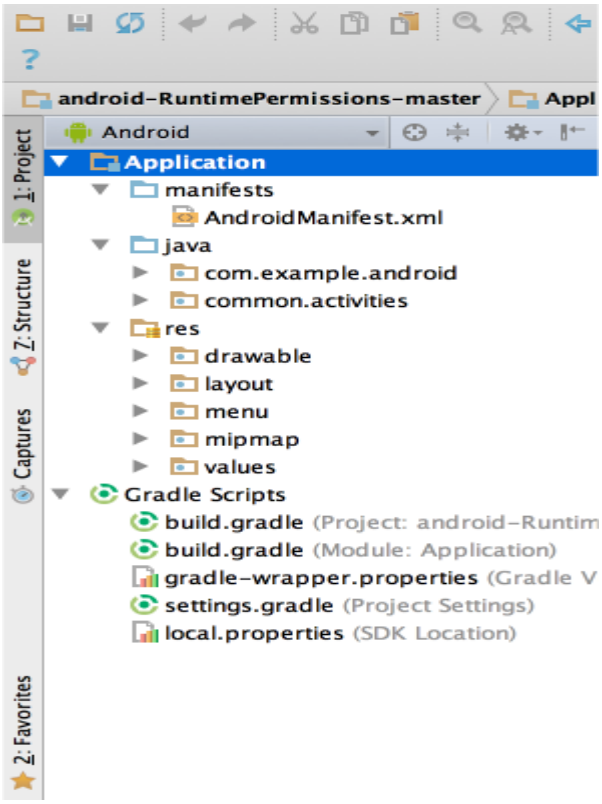
V osrednjem oknu se po prevzetem načinu odpre urejevalnik grafičnega vmesnika. Ob dvokliku na datoteko vrste Java se odpre urejevalnik kode za razvijanje v programskem jeziku Java (Slika 3.8).



Slika 3.8: Prikaz urejevalnika kode v Android studiu.

3.1.2 Struktura projekta

Struktura projekta je vidna v levem delu glavnega okna (Slika 3.9). V mapi Java so zbrane vse datoteke, ki upravljajo z logiko posameznih pogledov, v mapi res pa so zbrane datoteke, namenjene grafičnemu vmesniku aplikacije. Te datoteke so tipa XML in slike poljubnega tipa. Vse slike, uporabljene v aplikaciji, morajo biti zbrane v mapi res, zgolj na tak način se v okolju lahko nanje sklicujemo. Mapa ima privzeto že v naprej ustvarjene podmape s predpono minimap. Te so namenjene različnim resolucijam naprav (optimizacija). Cilj je optimiziran videz in pravilen prikaz slik. Določeno ozadje je večja slika kot slika, namenjena ikoni. Glede na napravo aplikacije za dano resolucijo ustrezno poišče pravo mapo in izbere določeno sliko. Če imamo slike zbrane samo v eni mapi, se bo aplikacija vedno sklicevala nanjo. Manifest mapa vsebuje samo eno datoteko tipa XML. V njej opredelimo povezave med datotekami Java in datotekami XML, namenjene grafičnemu prikazu. Prav tako na tem mestu nastavimo pravice, ki jih aplikacija potrebuje, in sicer dostop do funkcionalnosti mobilne naprave (kamera, splet in podobno). Pri tem koraku namreč potrebujemo soglasje uporabnika.



Slika 3.9: Prikaz strukture projekta v razvojem okolju.

3.1.3 Funkcionalnosti okolja

V tem poglavju bomo omenili vse glavne funkcionalnosti , ki nam jih nudi razvojno okolje Android studio [7].

- **Gradle orodje za gradnjo projekta**: Gre za sistem, ki datoteke združuje znotraj res in map Java ter jih prevede in zapakira v datoteko APK [11]. Datoteko v naslednjem koraku namestimo na naprave. Vse Android aplikacije so datoteke APK. Prej je razvijalec moral v ukazni vrstici pognati različna orodja, ki so ustvarila datoteko APK. Ta korak je zdaj neopazen. Ker okolje s tem upravlja samo, razvijalcu zanj ni potrebno skrbeti.

  Razvijalec lahko skripto Gradle orodja priredi. Gre za vtičniško zasnovan sistem (angleško plugin based system), kar pomeni, da lahko različne vtičnike (knjižnice) poljubno dodamo ali razvijemo. Gre za rešitev določenega problema, ki jo lahko javno objavimo in s tem pomagamo drugim razvijalcem.

- **Emulator**: Hitri in zmogljivi emulatorji so razviti s strani Googla. V emuatorju je pri izboru simulacije na voljo veliko število različnih mobilnih naprav.

- **Takojšnje prevajanje projekta (angleško instant run)**: Ko aplikacijo zaženemo ali razhroščujemo in smo v že aktivni aplikaciji spremenili del kode ali datoteke v mapi res, okolje pametno zazna spremembe in aplikacije ne zažene ponovno. Spremembe uspešno namesti in razvijalec lahko brez izgube časa ob ponovnem zagonu aplikacijo naprej uporablja.

- **Številne predloge za aplikacije z GitHuba.**

- **Vgrajena podpora za Google Cloud Platformo.**

- **Orodja za testiranje:** Podpira orodja JUnit 4 za testiranje kode in UI testna ogrodja (angleško frameworks).

- **C++ in NDK podpora:** Podpira za urejanje projektnih datotek, razvitih v programskem jeziku C ali C++ projektov. S tem lahko za naš projekt razvijemo komponente JNI (angleško Java Native Interface) [13]. Te se uporabljajo za izboljšanje performanc v programskem jeziku Java. Njihovo razvijanje je dokaj zapleteno in namenjeno izkušenejšim razvijalcem. Potrebno je poznavanje programskih jezikov C ali C++.

3.2 Xcode

Razvojno okolje Xcode [8] je osrednje okolje za razvoj aplikacij, ki delujejo na Apple napravah. Trenutna različica okolja je Xcode 8. Okolje je integrirano z ogrodji Cocoa in Cocoa Touch [4]. Ogrodje Cocoa je namenjeno razvoju namiznih aplikacij, Cocoa Touch pa razvoju aplikacij na platformi iOS. Hitro opazna razlika med ogrodji so različno poimenovani razredi. V ogrodju Cocoa imajo vsi razredi predpono NS. Nasprotno se imena razredov v ogrodju Cocoa Touch začnejo s predpono UI. Obe ogrodji se povezujeta z ogrodjem Foundation, ki je osnova. To ogrodje ima izboljšano uporabnost za arhitekturni MVC (poglavje 2.2) pristop razvoja aplikacije. V okolju lahko razvijemo aplikacije, ki so namenjene:

- osebnim računalnikom Mac (OSX),

- mobilnim napravam iPhone (iOS),

- tabličnim računalnikom iPad,

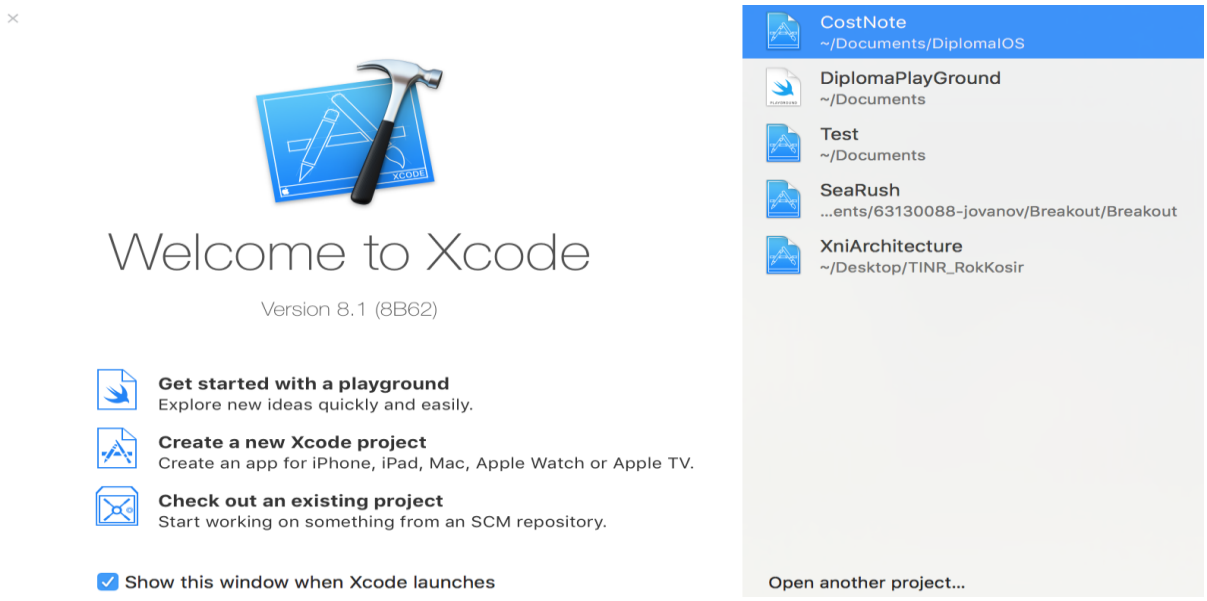
- pametnim uram Apple Watch,

- pametnim televizijam Apple TV.

Okolje Xcode podpira razvoj v programskih jezikih Objective-C in Swift. Trenutna različica iOS je različica 10. Ob namestitvi okolja Xcode različice 8 se iOS 10 prenese kot privzeti iPhone SKD. Tako kot okolje Android studio tudi Xcode podpira hitre in zmogljive emulatorje mobilnih naprav. Apple naprav je veliko manj kot Android, zaradi česar so naprave in iOS različice bolj optimizirane.

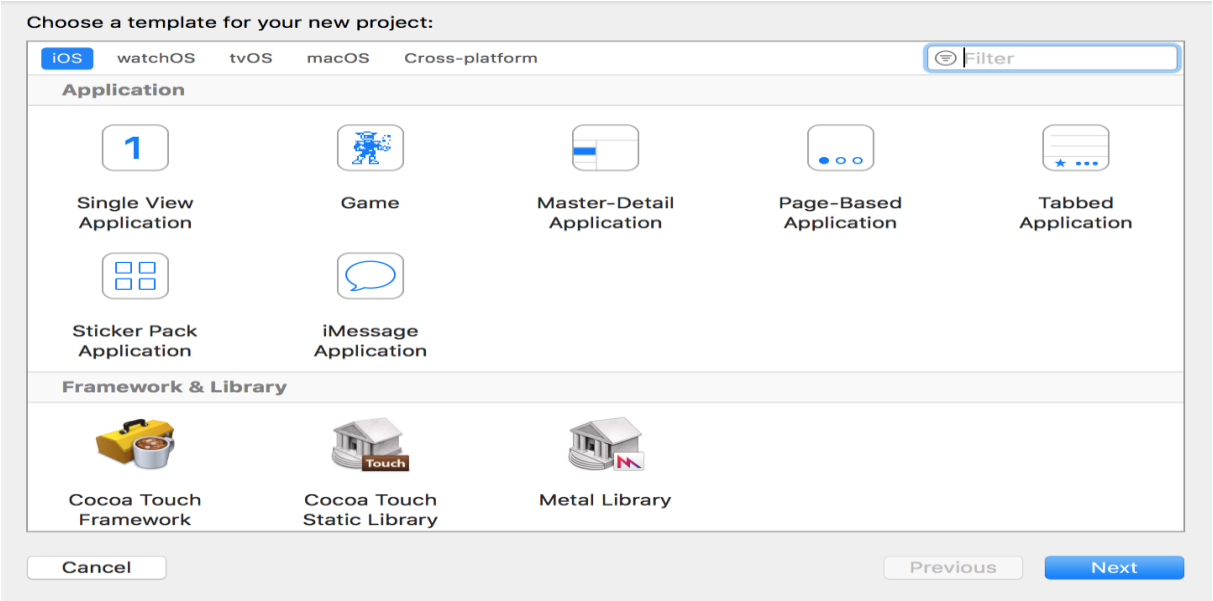
3.2.1 Kreiranje novega projekta

Videz grafičnega vmesnika okolja se razlikuje od okolja Android studia, osnovni koncept pa je podoben. Okolje Xcode razvijalca prav tako vodi čez vse korake ter skrbi za konsistentnost in dobro uporabniško izkušnjo. V začetnem pogledu okolja (Slika 3.10) lahko izberemo ustvarjenje novega projekta ali urejanje že obstoječega.



Slika 3.10: Začetni pogled okolja Xcode.

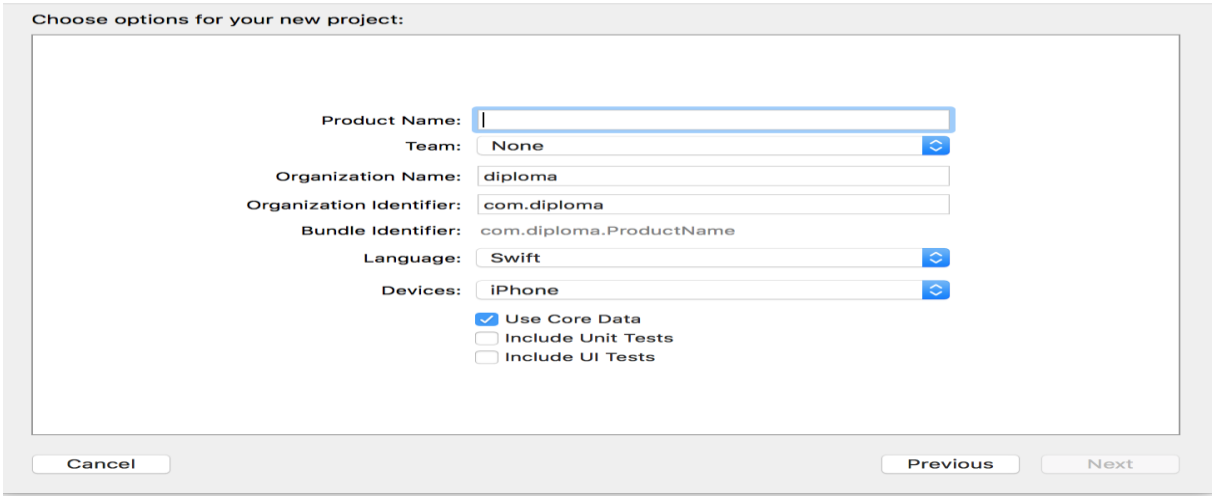
Ob izbiri novega projekta nas okolje vodi v naslednji korak (Slika 3.11), pri katerem izberemo vrsto in predlogo nove aplikacije. Največja razlika je uporaba navigacije za prehajanje med pogledi. Ponuja možnost razvoja mobilnih iger in klepetalnika. V diplomski nalogi smo izbrali posamezno pogledno vrsto (Single view application), ker predlog nismo potrebovali in smo za navigacijo poskrbeli sami.



Slika 3.11: Izbira vrste aplikacije.

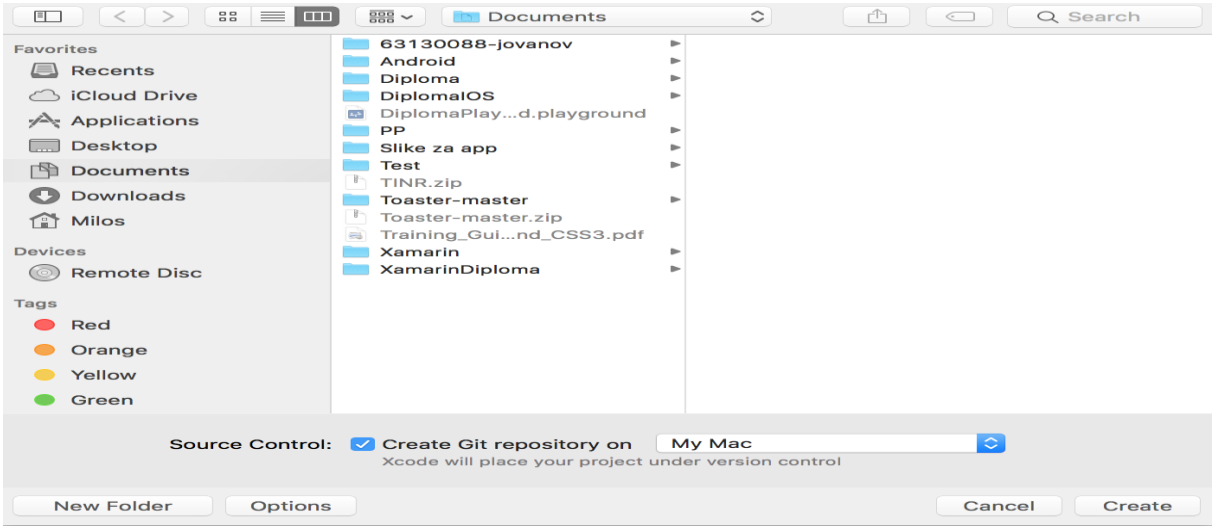
V naslednjem koraku sledi poimenovanje projekta in izbira naprave, za katero želimo razviti aplikacijo (Slika 3.12). Prav tako izberemo poljubni programski jezik razvoja in hkrati vključimo tudi možnost uporabe ogrodja Core data, ki skrbi za upravljanje s podatkovno bazo SQLite in je privzeto nameščeno v okolju Xcode. Ogrodje lahko v naš projekt vključimo tudi kasneje. Več o tem bomo podrobneje opisal v naslednjem poglavju.

Potrebno je omeniti tudi izbiro razvijalske ekipe, s pomočjo katere lahko aplikacijo namestimo na fizično napravo. Pomembna je za podpis (certifikat) aplikacije, brez katere je na fizično napravo ni mogoče namestiti.



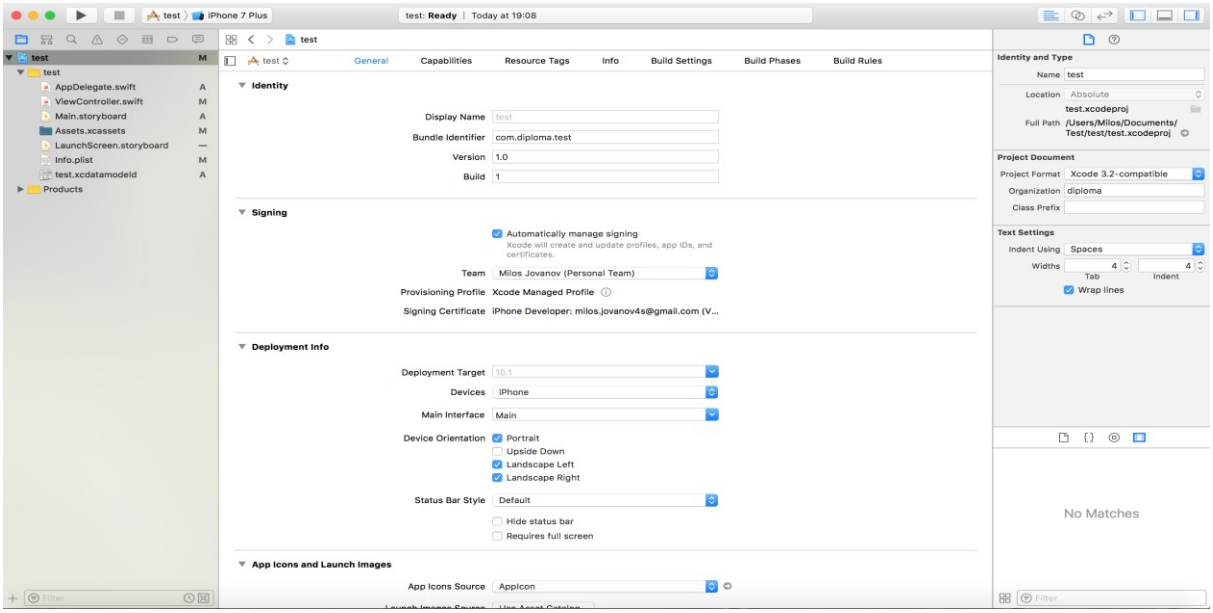
Slika 3.12: Poimenovanje projekta, izbira ciljne naprave in programskega jezika.

Sledi izbira lokacije projekta na disku osebnega računalnika (Slika 3.13).



Slika 3.13: Izbira lokacije shranjevanja projekta.

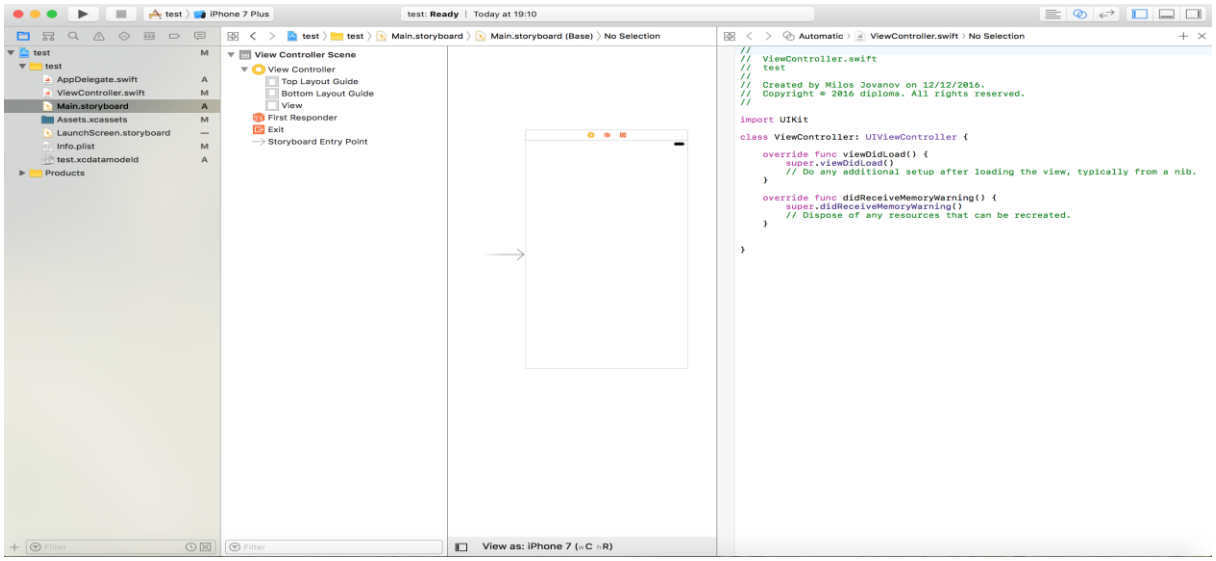
Okolje v naslednjem koraku odpre glavno okno projektnih nastavitev (Slika 3.14). Določimo lahko ciljno iOS različico in podpise, dodamo zunanje knjižnice ter druge lastnosti projekta.



Slika 3.14: Osrednje okno za določanje lastnosti in konfiguracijo projekta.

V primeru izbire obstoječega projekta okolje odpre glavno okno z urejevalnikom pogledov in urejevalnikom kode (Slika 3.15). Razvojno okolje Xcode ima zelo dobro podporo za prilagajanje okolja po željah razvijalca. Pri razvijalcih je cenjena funkcionalnost hitrega

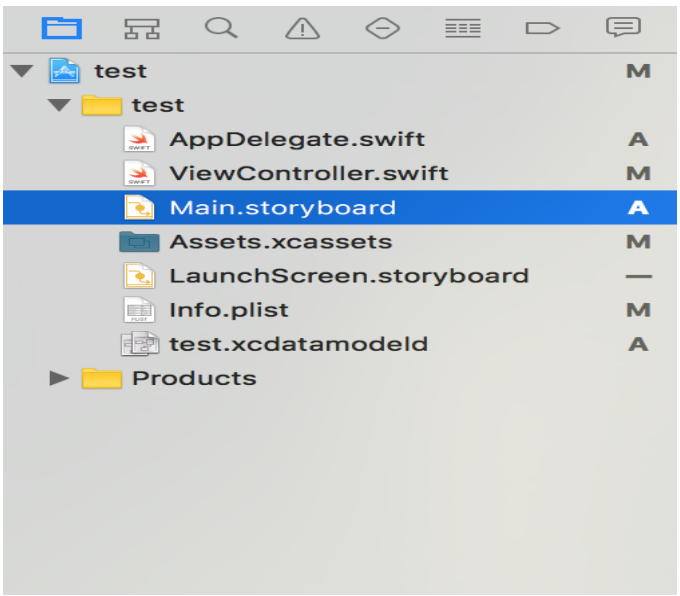
preklopa med urejevalniki. Ob izbiri različne vrste datoteke se dinamično odpira ustrezni urejevalnik (urejevalnik za grafični vmesnik ali urejevalnik kode).



Slika 3.15: Glavno okno z urejevalnikom pogledov in urejevalnikom kode.

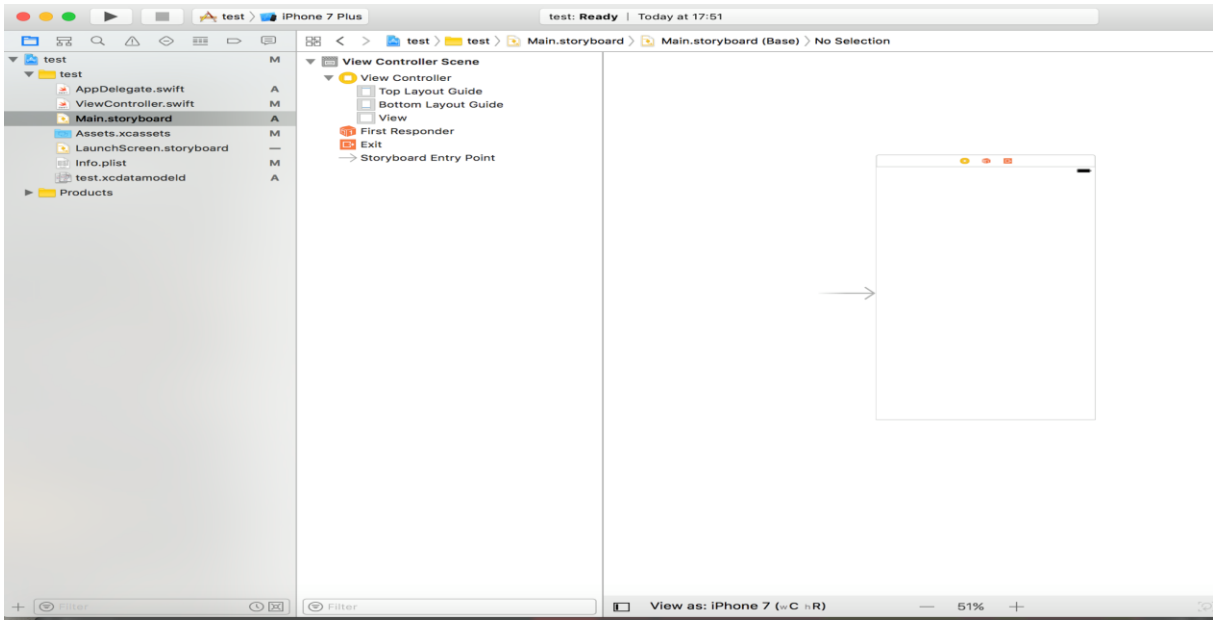
3.2.2 Struktura projekta

Drevesna struktura projekta je na levi strani glavnega okna (Slika 3.16). Po prevzetem načinu projekt ni strukturiran po mapah kot pri Android studiu. Vse datoteke so zbrane v osrednji mapi projekta. Končna aplikacija, ki jo okolje prevede za napravo, je v mapi Products. Prevedena aplikacija je končnice app.



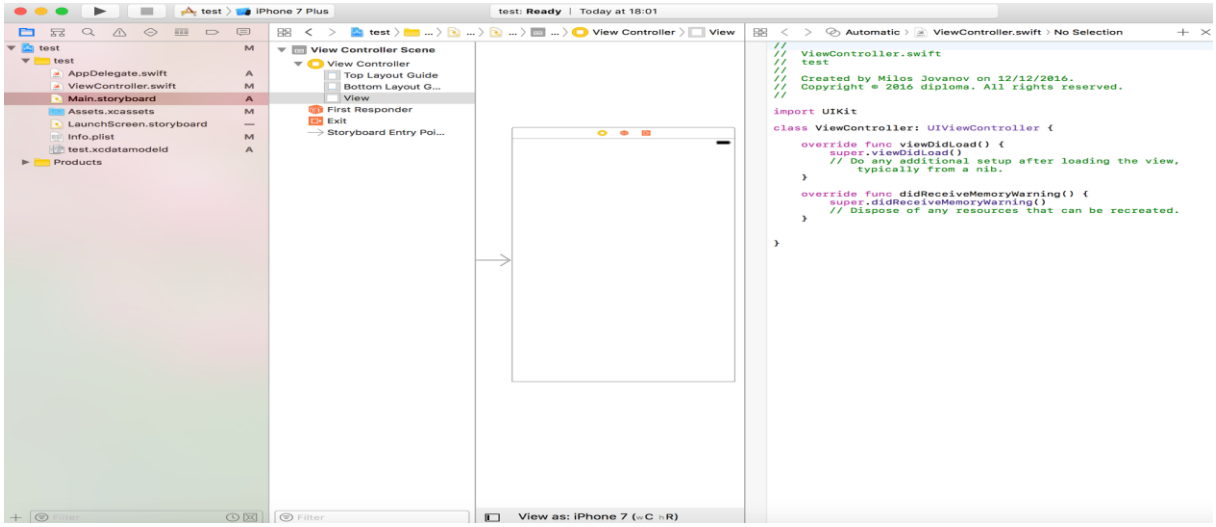
Slika 3.16: Struktura projekta v Xcode.

Razvijalec lahko poljubno dodaja mape in v njih prestavlja datoteke ter ustvari poljubno strukturo projekta. Poljubno dodajanje map je možno tudi v Android studiu. Datoteke vrste swift skrbijo za logiko posameznega pogleda in so krmilniki v arhitekturnem pogledu MVC. Za razvijanje grafičnega vmesnika okolje Xcode uporablja datoteko vrste Storyboard. Z izbiro te datoteke v strukturi projekta okolje odpre grafični vmesnik za razvijanje videza aplikacije (Slika 3.17). V urejevalniku lahko razvijalec z metodo povleci in spusti razvije videz aplikacije. Preprosto izbere komponento in jo postavi na želeno mesto. V datoteki Storyboard lahko razvijalec prav tako določi potek delovanja aplikacije, ki je namenjen navigaciji med pogledi (okna aplikacije).



Slika 3.17: Preusmeritev v urejevalnik grafičnega vmesnika aplikacije.

Okolje ponuja številne predloge za navigacijo med pogledi. Podprte navigacije delujejo na osnovi definicij za potek prehodov med pogledi. V projektu diplomske naloge smo navigacijo implementirali sami. Potek aplikacije smo prav tako določili v datoteki Storyboard. S pomočjo urejevalnika asistenta imamo lahko v osrednjem oknu aplikacije sočasno odprto tudi pripadajočo swift datoteko (krmilnik pogleda). To omogoča sočasen razvoj grafičnega vmesnika in logičnega dela aplikacije (Slika 3.18).



Slika 3.18: Na desni strani je viden urejevalnik asistent.

Urejevalnika sta tesno povezana in omogočata preprosto povezovanje komponent grafičnega vmesnika aplikacije in njenega zaledja (logika). To je ena boljših funkcionalnosti okolja, ki vidno pohitri razvoj aplikacije. Mapa Assets je namenjena slikam, ki jih uporablja aplikacija. Okolje ponuja shrambo istoimenske slike v različnih velikostih (resolucija). Glede na uporabo aplikacije na napravi izbere primerno velikost slike. Datoteka vrste plist je sistemska datoteka aplikacije, hrani pa nastavitve in pravice. V datoteki vrste plist smo dodali pravice za uporabo kamere na mobilni napravi. Za upravljanje s podatki uporablja orodje CoreData datoteko xcdatamodel. Razvijalec na tem mestu ustvari model podatkovne baze, ki se preslika v podatkovno bazo SQLite. Za razvijanje modela ima okolje Xcode grafični vmesnik in ni potrebno pisanje SQL poizvedb.

3.2.3 Funkcionalnosti okolja

V tem poglavju bomo predstavili funkcionalnosti, ki zaznamujejo okolje Xcode [9].

- **Integriran sistem za gradnjo aplikacije**: Zmogljiv sistem, ki projekt prevede v datoteko vrste app. Ta se namesti na emulatorju ali napravi. Sistem poskrbi za podpis aplikacije (certifikat).

- **iOS emulator**: Z iOS SDK se namesti tudi ustrezen emulator. Njegovo delovanje je možno samo na osebnem računalniku z operacijskim sistemom OSX.

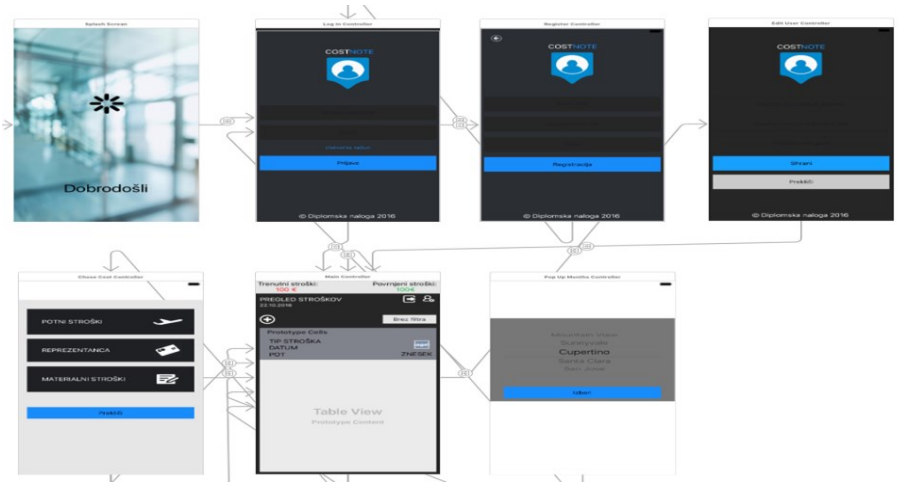
- **Urejevalnik pomočnik** (angleško Assitant Editor): Z izbiro pomočnika (gumb v orodni vrstici) se v glavnem oknu okolja prikaže dodatni urejevalnik. V njem se prikaže vsebina datoteke, ki je po oceni okolja Xcode za razvijalca v danem trenutku najbolj uporabna.

Primer je lahko urejanje pogleda v grafičnem urejevalniku, v pomočniku se prikaže krmilnik pogleda (swift datoteka). Če razvijalcu izbor ni všeč, lahko v urejevalniku pomočniku sam izbere drugo vidno datoteko.

- **Urejevalnik slik** (angleško Asset Catalog): Urejevalnik slik je namenjen upravljanju uporabljenih slik v aplikaciji. Enako sliko združi v sklop slik z različnimi resolucijami in ga pri gradnji aplikacije učinkovito doda. Glede na velikost mobilne naprave se prikazuje ustrezna slika.

- **XCTest ogrodje za testiranje aplikacij** (angleško XCTest Framework): To ogrodje omogoča razvoj unit testov. Delujejo na napravah iPhone in iPad, osebnem računalniku Mac ter iOS emulatorju.

- **Urejevalnik vmesnika**: Ogrodje Cocoa Touch je zasnovano na arhitekturnem pristopu MVC. Razvoj grafičnega vmesnika in logike aplikacije sta lahko povsem ločena procesa. Urejevalnik je hiter in prilagodljiv ter nudi prijetno uporabniško izkušnjo. V povezavi z urejevalnikom pomočnikom omogoča okolje Xcode razvoj celotne aplikacije z metodo povleci in spusti (angleško drag & drop). Dodana vrednost urejevalnika so tudi Storyboardy, katerega osnovna ideja je določanje prehodov med pogledi. Z njihovo pomočjo nato implementiramo navigacijo aplikacije. Prehodi predstavljajo razmerja med pogledi in so v urejevalniku prikazani kot puščice (Slika 3.19). Njihova orientacija oziroma smer določa, iz katerega pogleda se dostopa do željenega pogleda in kateri pogled je naslednji. Puščice lahko služijo tudi kot konceptualni model aplikacije za prikaz delovanja naročniku/stranki.



Slika 3.19: Sive puščice predstavljajo Storyboarde.

3.3 Xamarin studio

Razvojno okolje Xamarin studio [10] omogoča razvoj aplikacij Xamarin.iOS, Xamarin.Mac, Xamarin.Android in Xamarin.Forms. Okolje je namenjeno operacijskemu sistemu OSX. Ne glede na izbiro tipa ciljne aplikacije se vse podprte platforme razvijajo v programskem jeziku C#. Xamarin je bila pred leti samostojna platforma, namenjena večplatformskemu (angleško cross-platform) razvoju mobilnih aplikacij. Večplatformski razvoj je razvijanje ene aplikacije, ki deluje na več platformah hkrati (primer sta lahko iOS in Andorid). Aplikacije ni potrebno razvijati za vsako platformo posebej. Pod svoje okrilje jo je vzel Microsoft in leta 2013 v sodelovanju s platformo Xamarin razvil Xamarin studio. Razvoj aplikacij Xamarin je možen tudi v Microsoftovem okolju Visual Studio. Namesti se kot dodatek in deluje znotraj okolja. Poenoten programski jezik C# je možen zaradi odprtokodnega okolja Mono, na katerem temeljita Xamarin.iOS in Xamarin.Android. Razvojno okolje Mono je kompatibilno z vsemi vodilnimi operacijskimi sistemi osebnih računalnikov (MS Windows, Mac OSX, Linux), igralnimi konzolami (Sony PlayStation, Nintendo Wii) in mobilnimi napravami (iOS in Android). V Visual Studiu z dodatkom Xamarina je možen tudi razvoj mobilnih aplikacij za Windows Phone, česar Xamarin studio ne omogoča. Aplikacija diplomske naloge je bila razvita na platformi Xamarin.Forms.

Kratek opis platform znotraj Xamarin studia:

- **Xamarin.iOS**: Na platformi poteka razvoj domorodnih mobilnih aplikacij iOS. Omogočen je grafični urejevalnik za razvoj vmesnika.

- **Xamarin.Android**: Na platformi poteka razvoj domorodnih Android mobilnih aplikacij. Omogočen je grafični urejevalnik za razvoj vmesnika.

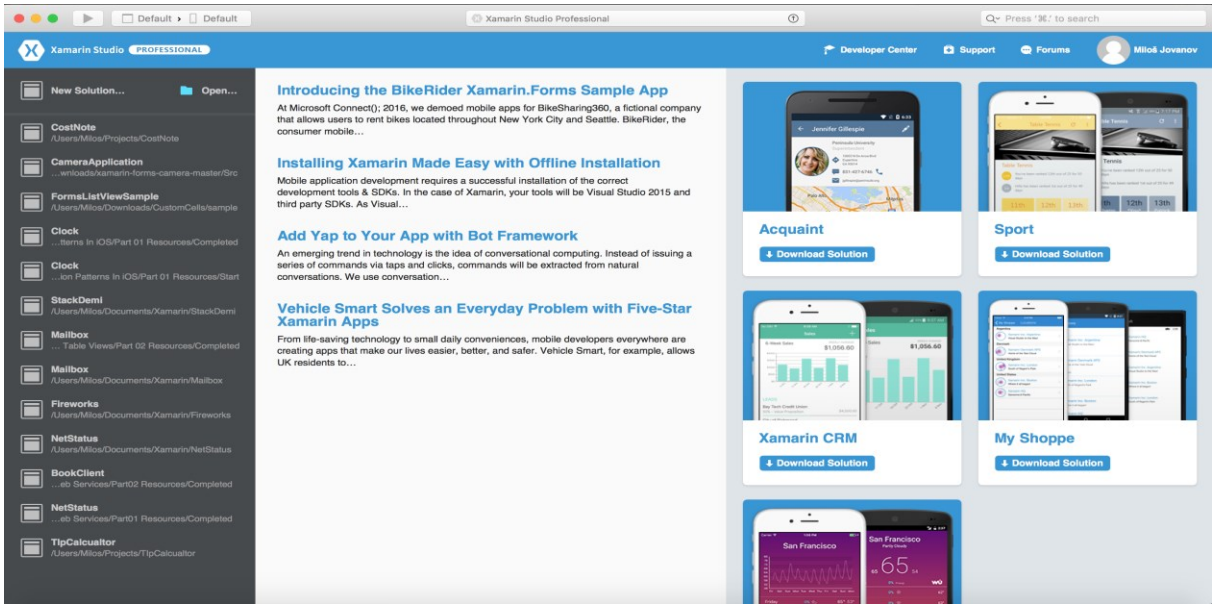
- **Xamarin.Mac**: Na platformi poteka razvoj namiznih aplikacij za operacijske sisteme OSX. Omogočen je grafični urejevalnik za razvoj vmesnika.

- **Xamarin.Forms**: Na platformi poteka razvoj večplatformskega razvoja mobilnih aplikacij Android in iOS. Večplatformski razvoj omogoča deljeno kodo in skupni projekt. Končna aplikacija se prevede na iOS in Android platformo. Grafičnega urejevalnika za razvoj vmesnika ni. Celotni razvoj grafičnega vmesnika aplikacije poteka v formatu XAML in programskem jeziku C#. Razlog je razlika komponent med platformama iOS in Android, uporabljenih za razvoj grafičnega vmesnika. Vsaka komponenta, opisana v datoteki XAML, se ustrezno preslika v komponento domorodne aplikacije. Urejevalnik grafičnega vmesnika za Xamarin.Forms je v fazi razvoja in bo v prihodnosti tudi tu omogočen. Projekt Xamarin.Forms je sestavljen iz treh

podprojektov. Glavni projekt je projekt deljene kode. V njem razvijalec razvije vse potrebne poglede in njihove krmilnike, ki skrbijo za logiko. Ostala projekta sta projekta iOS in Android. V njiju podpremo vse posebnosti specifične platforme (iOS ali Android). Te posebnosti so lahko videz in obnašanje komponent grafičnega vmesnika, ki se med platformami razlikujejo. Logika aplikacije je v projektu z deljeno kodo in podprojekta se zgolj sklicujeta nanj. Ko želimo aplikacijo namestiti, moramo določiti glavni projekt za gradnjo. Hkrati je lahko izbran bodisi iOS projekt bodisi Android projekt. Sistem za gradnjo nato deljeno kodo prevede v izbran projekt in njegovo platformo.

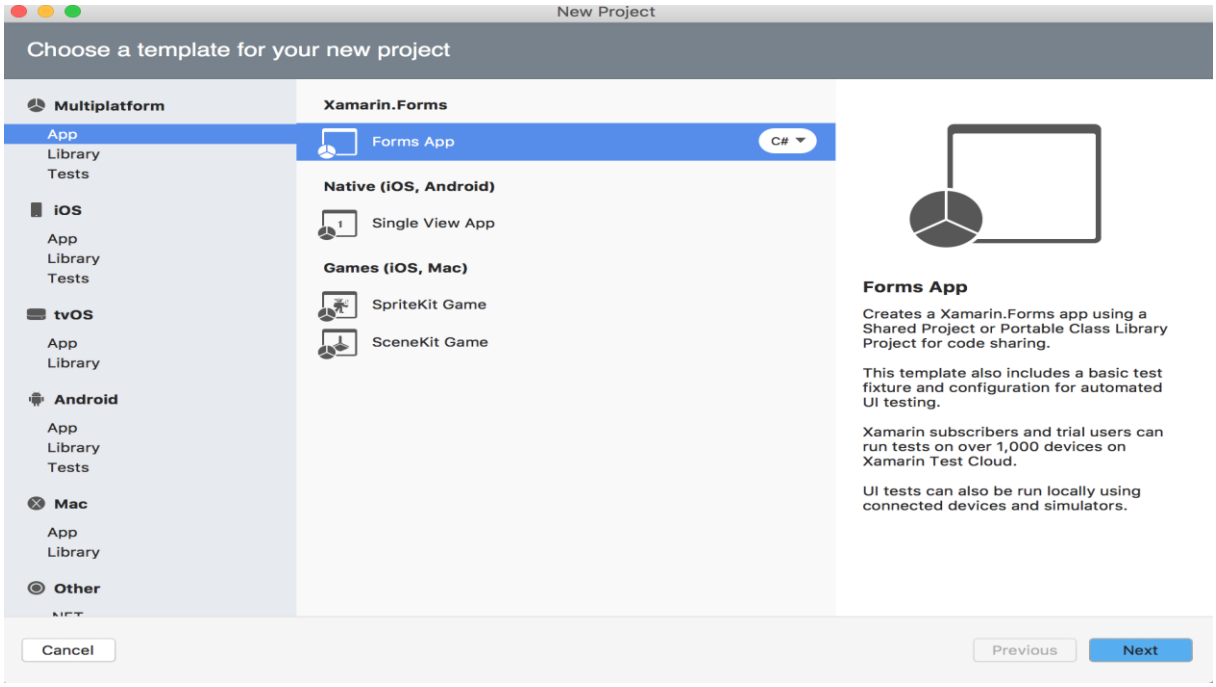
3.3.1 Kreiranje novega projekta

V tem poglavju bomo opisali še zadnje uporabljeno razvojno okolje diplomske naloge. Tako kot njegova predhodnika Xamarin studio skrbi za vodeno ustvarjanje novega projekta. Ena od razlik je vidna že v prvem oknu razvojnega okolja Xamarin studio (Slika 3.20).



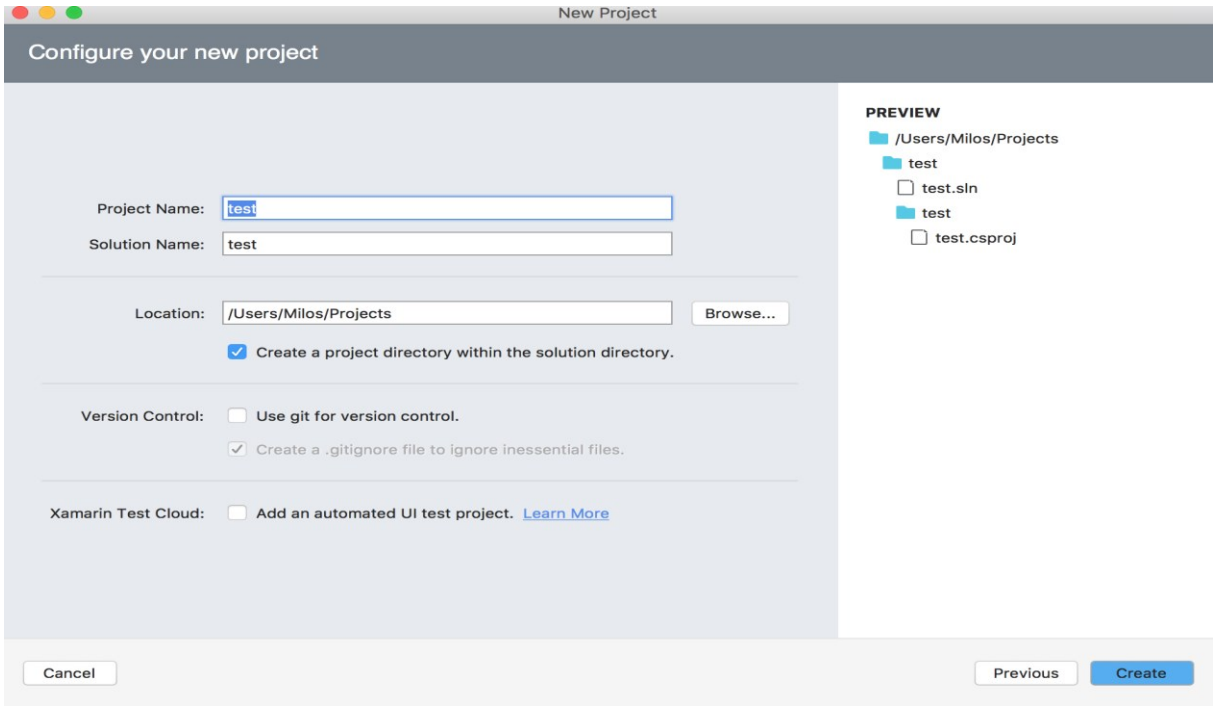
Slika 3.20: Začetno okno razvojnega okolja Xamarin studio.

Na desni strani lahko razvijalec izbere že obstoječi projekt ali ustvari novega. V primeru izbire obstoječega projekta se odpre osrednje okno okolja, nasprotno pa okno za izbiro vrste aplikacije (Slika 3.21).



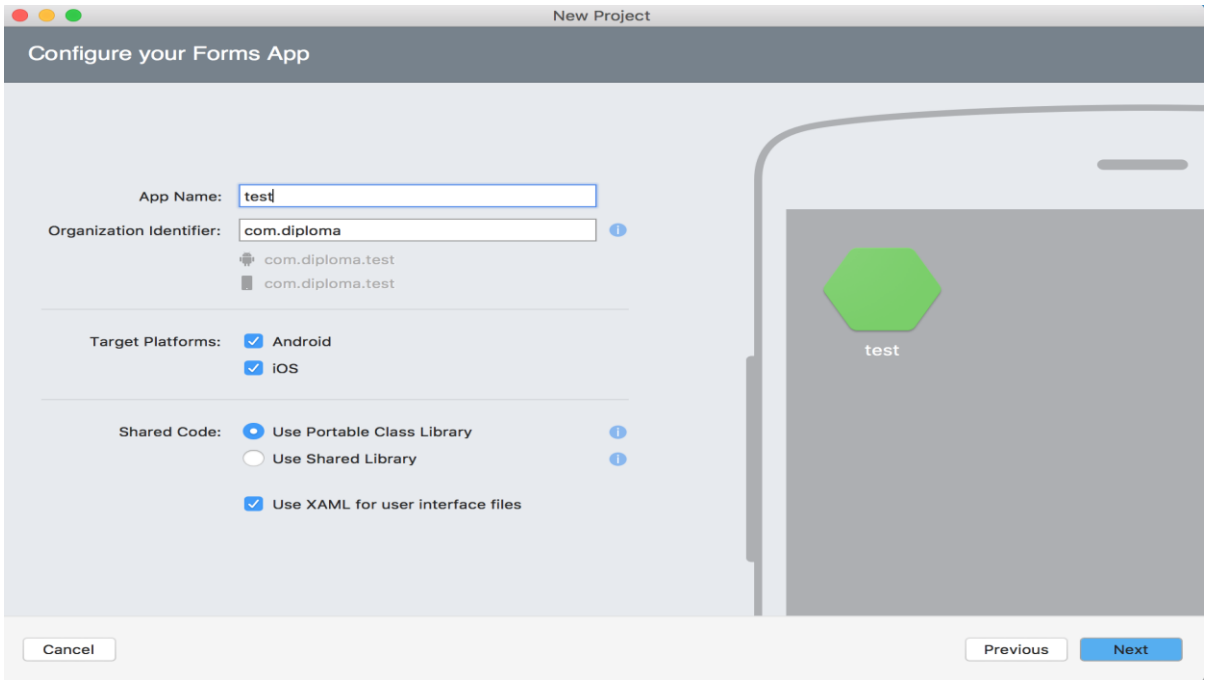
Slika 3.21: Okno za izbiro vrsto aplikacije.

V tem oknu vidimo vse zgoraj omenjene vrste, ki jih podpira razvojno okolje. Pod rubriko Multiplatform je naša ciljna izbira Xamarin.Forms (Forms App). Sledi korak poimenovanja novega projekta (Slika 3.22).



Slika 3.22: Poimenovanje projekta in izbira lokacije shrambe na računalniku.

Desno v oknu je viden pregled potencialne strukture projekta. V naslednjem koraku določimo ciljne platforme in način implementacije deljene kode med projekti (Slika 3.23). Na izbiro imamo Portable Class Libary (PCL) in Shared Libary (Shared project).



Slika 3.23: Izbira ciljnih platform.

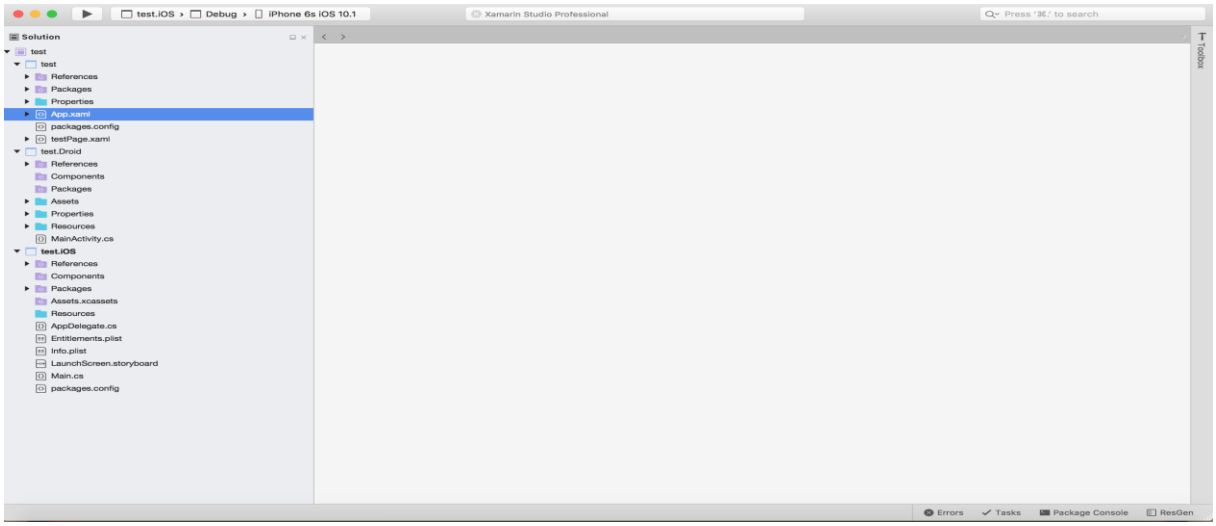
Shared project se uporablja za zelo preproste aplikacije. Njegova slabost je, da mora razvijalec ob sklicevanju na specifično lastnost platforme podpreti vsako platformo posebej (Slika 3.24). Koda tako lahko postane slabo berljiva in kompleksna za vzdrževanje, saj mora razvijalec na več mestih popraviti željeno spremembo.

```
#if WINDOWS_PHONE
path = "windowsphone";
#else

#if __SILVERLIGHT__
path = "silverlight";
#else

#if __ANDROID__
path = "android";
#else

#if __IOS__
path = "iOS";
#else
```

Slika 3.24: Primer podprtja vseh željenih platform.

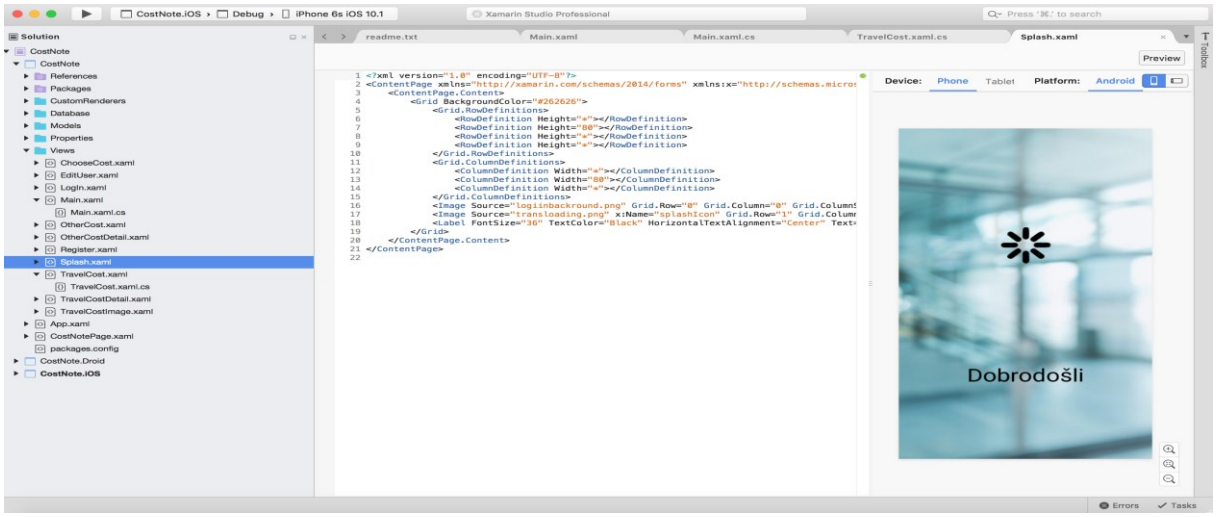
V praksi se pogosteje uporablja vrsta projekta PCL. Njihova prednost je enotna koda vseh logik, pri kateri vrste implementacije glede na željeno vrsto platforme ni potrebno opredeliti. Specifične lastnosti platform se implementira v podprojektu platforme. V projektu deljene kode razvijalec opredeli posamezni vmesnik. Podprojekti platform vmesnik dedujejo in specifične lastnosti implementiramo glede na potrebe platforme.

Xamarin studio v naslednjem koraku odpre svoje osrednje okno (Slika 3.25). Struktura je vidna na levi strani okna, ustrezni urejevalnik pa ob izbiri datoteke zasede desni del osrednjega okna okolja.



Slika 3.25: Glavno okno Xamarin studia.

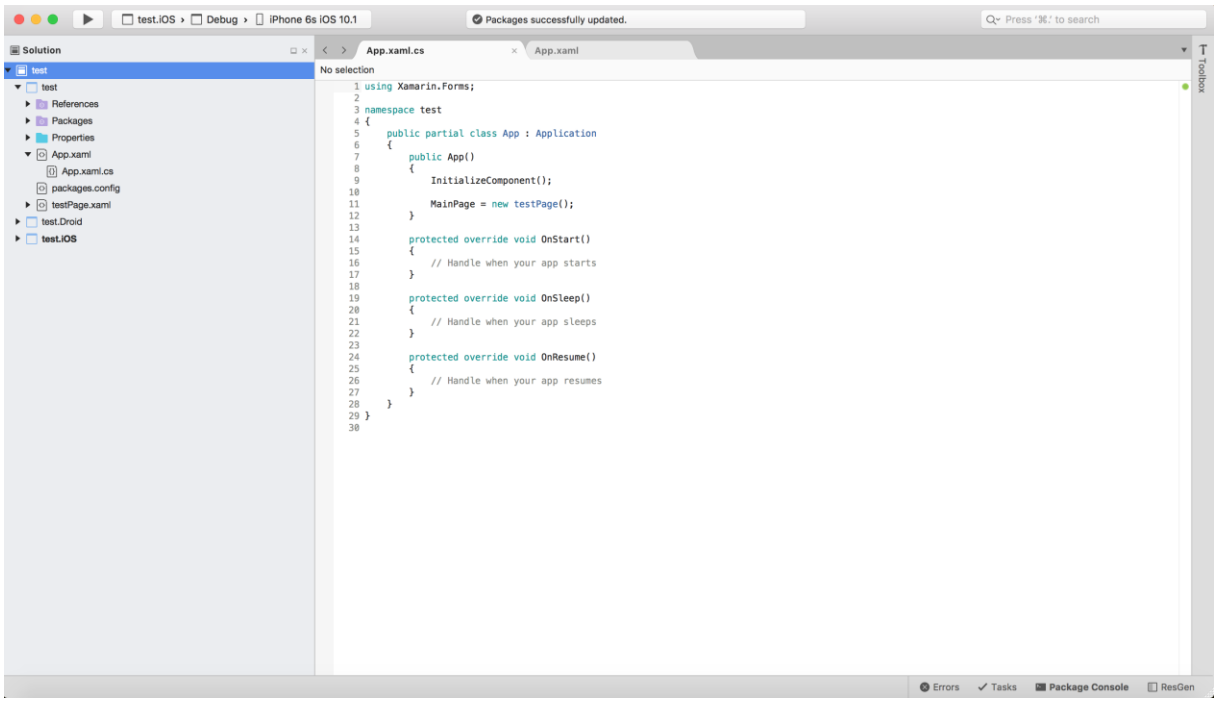
Ob dvokliku datoteke XAML se odpre urejevalnik grafičnega vmesnika (Slika 3.26).



Slika 3.26: Urejevalnik grafičnega vmesnika aplikacije.

Desno okno prikazuje videz vmesnika na mobilni napravi. Celoten videz aplikacije se razvija v datoteki XAML brez vmesnika, ki bi omogočal funkcijo vleci in spusti (angleško drang & drop) komponent.

Ob dvokliku na datoteko s končnico cs se odpre urejevalnik kode v programskem jeziku C# (Slika 3.27).



Slika 3.27: Urejevalnik kode v Xamarin studiu.

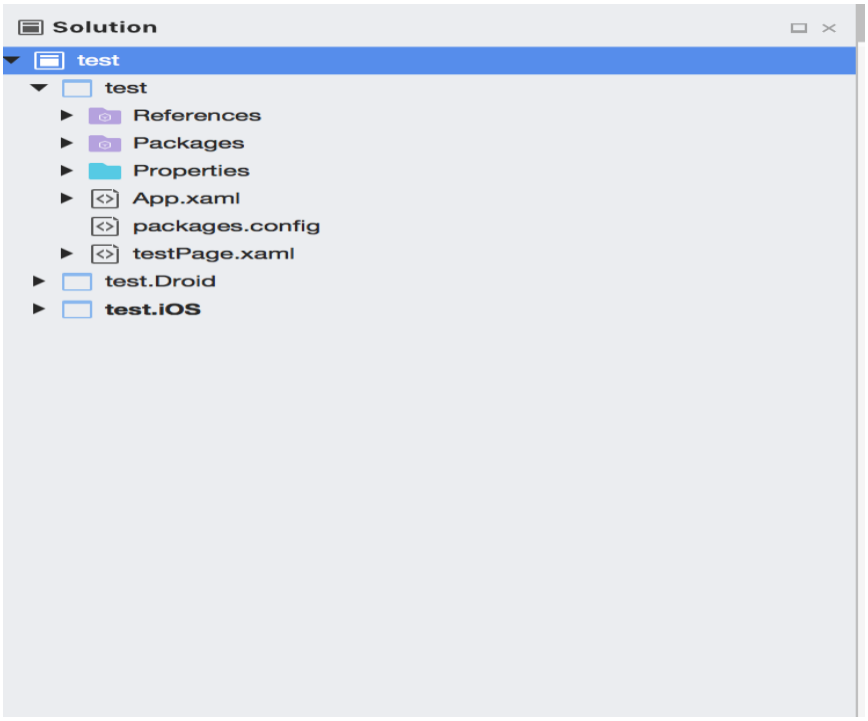
3.3.2 Struktura projekta

Struktura projekta je v levem delu razvojnega okolja (Slika 3.28). Posebnost strukture v projektu z deljeno kodo je, da vsebuje tri podprojekte, in sicer:

- Projekt z deljeno kodo.
- Projekt Android (končnica .Android).
- Projekt iOS (končnica .iOS).

Projekt z deljeno kodo je osrčje večplatformskega razvoja v Xamarin.Forms. Razvijalec v njem razvije vso logiko in videz aplikacije. S potrebo na sklicevanje specifik določene platforme se implementira in podpre v željenem podprojektu (iOS ali Android). Posebnost tega okolja je izbira projekta za prevajanje in gradnjo aplikacije. Na sliki (Slika 3.28) je kot osrednji projekt

izbran iOS (črno poudarjeno ime), kar pomeni, da se bo aplikacija namestila na iOS emulatoju ali fizični mobilni napravi s platformo iOS. Razvijalec lahko vsak trenutek izbere glavni projekt in ponovno izvede prevajanje in gradnjo aplikacije.



Slika 3.28: Struktura projekta v Xamarin studiu.

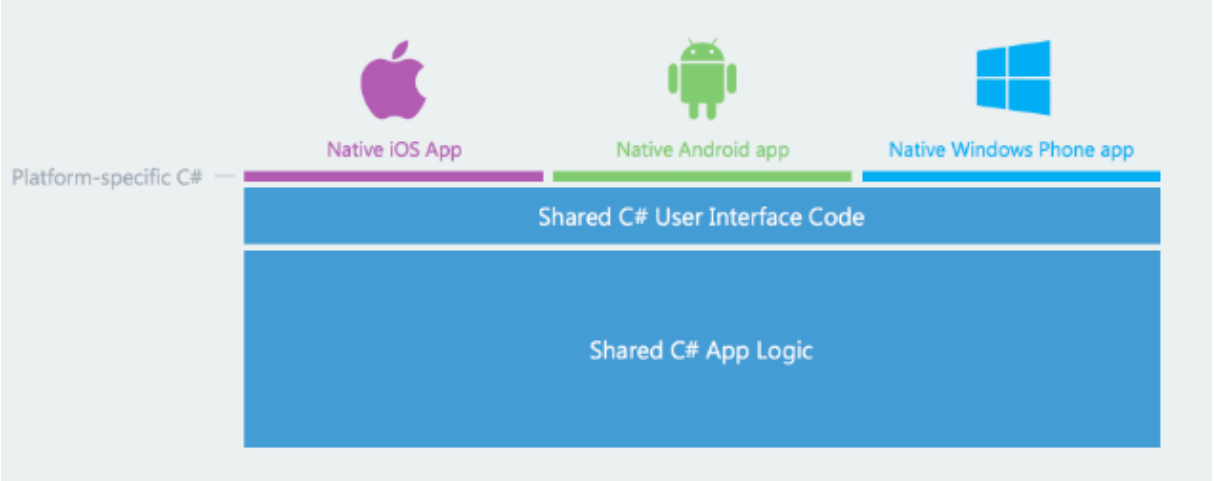
3.3.3 Funkcionalnosti okolja

Namen tega poglavja je predstavitev glavnih funkcionalnosti [23] razvojnega okolja Xamarin studio.

- **Poenoten programski jezik C#**: Vse podprte platforme so razvite v enem programskem jeziku (C#), razvijalci se tako lahko osredotočijo na posamezen programski jezik. Produktivnost in hitrost razvoja se z izkušnjami lahko znatno izboljšajo.

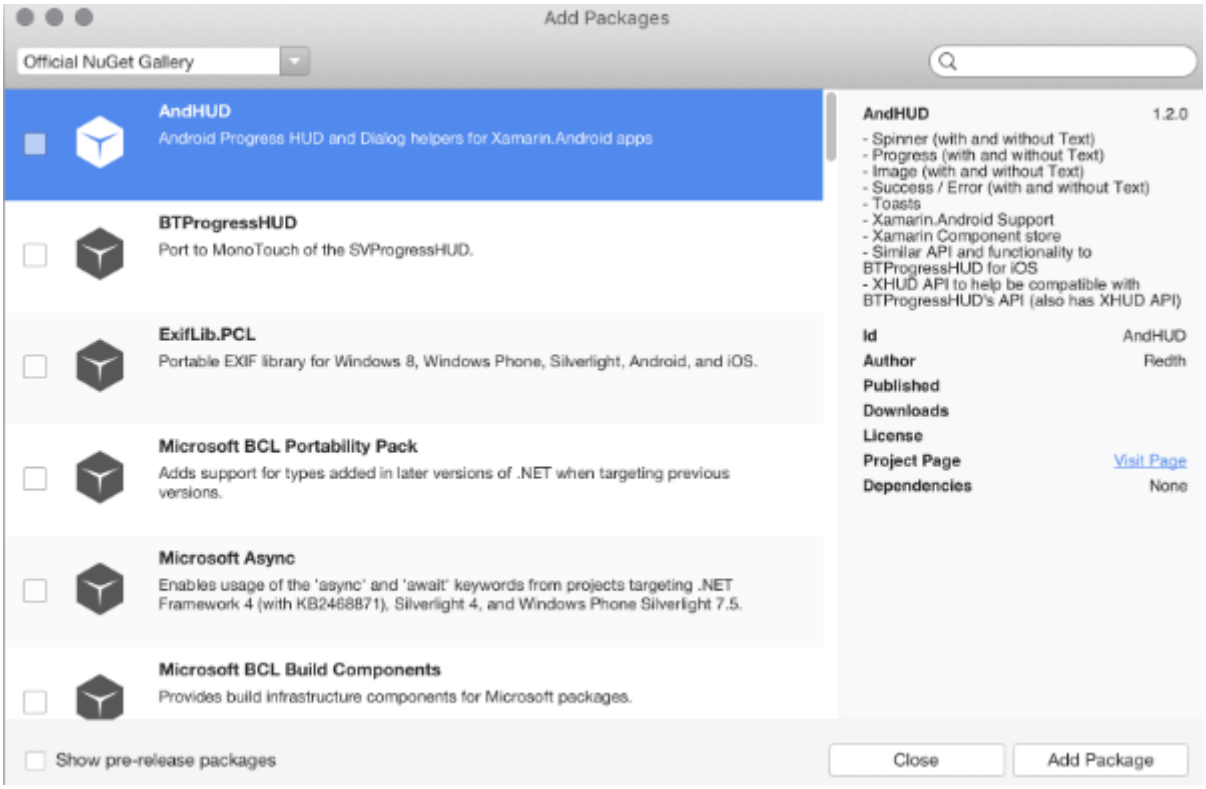
- **Urejevalniki grafičnega vmesnika**: Razvoj domorodnih aplikacij je podprt z urejevalnikom grafičnega vmesnika (iOS in Android), ki podpira vse zahteve specifične platforme. Za razvoj večplatformske aplikacije z deljeno kodo grafični vmesnik za razvijanje videza aplikacije trenutno ne obstaja.

- **Deljenje kode med platformami**: Koda se lahko med različni platformami deli tudi do 75%, z uporabo vrste projekta Xamarin.Forms pa se ta odstotek približa kar 100%. Koncept deljene kode med platformami je viden na spodnji sliki (Slika 3.29).



Slika 3.29: Koncept deljenja kode na platformi Xamrin.

- **Nuget packet manager**: Nuget je čarovnik (Slika 3.30) za dodajanje, posodabljanje ali odstranjevanje poljubnih knjižnic, ki uporabljajo ogrodje .NET. Knjižnice v iskalniku Nuget imajo uradno dokumentacijo in so zanesljive. Trenutno je podprtih preko 22 tisoč knjižnic in številka tedensko raste.



Slika 3.30: Čarovnik Nuget za dodajanje knjižnic.

Poglavje 4 Primerjava razvoja aplikacije v različnih razvojnih okoljih

V poglavju bomo primerjali razvoj aplikacije Cost Note, ki je potekal v vseh, v predhodnjih poglavjih omenjenih, razvojnih okoljih. Za dobro primerjavo je potrebno določiti primerjalne kriterije. Na osnovi tega bomo prikazali razlike, prednosti in slabosti posameznih razvojnih okolij. Izvedli smo naslednje primerjave razvojnih okolij:

- Primerjava razlik komponent za gradnjo grafičnega vmesnika.
- Primerjava razlik končnega videza grafičnega vmesnika aplikacije med domorodnima aplikacijama in aplikacijo Xamarin.Forms.
- Primerjava uporabe podatkovne baze SQLite.
- Primerjava hitrosti razvoja aplikacije med razvojnimi okolji.
- Uporabljene zunanje knjižnice za razvoj aplikacije.
- Povezovanje grafičnega vmesnika (pogleda) z logiko (krmilnik).
- Primerjava implementacij glavnih funkcionalnosti in posebnosti v posameznem okolju.

Vsa primerjana okolja imajo uradno dokumentacijo dostopno na spletu. Pri razvoju so bile v veliko pomoč. Vsebujejo širok nabor primerov in uporabe vseh funkcionalnosti komponent, ki jih razvijalec lahko uporabi pri razvoju poljubne aplikacije.

4.1 Primerjava komponent za razvoj grafičnega vmesnika

V poglavju bomo predstavili komponente, s pomočjo katerih smo razvili željen grafični vmesnik aplikacije. Prikazali bomo videz komponent, uporabljene lastnosti, imena in razrede, uporabljene v programskih jezikih razvojnega okolja. Komponente uporabniku omogočajo bodisi upravljanje z aplikacijo, bodisi služijo kot prikaz določenega podatka. Lastnosti

ponavljajočih se komponent bodo pojasnjene samo ob prvi omembi. To velja za vsako okolje posebej. Lastnosti smo v okoljih Android studio in Xcode lahko urejali s pomočjo grafičnega vmesnika. Okolje Xamarin studio trenutno ne podpira grafičnega vmesnika, zato smo lastnosti urejali v urejevalniku XAML.

4.1.1 Vnosno polje

Komponenta vnosno polje uporabniku služi za vnos željenega podatka. Lahko je v obliki besedila, številke ali obojega. Na spodnji sliki (Slika 4.1) so vidne osnovne razlike vnosnih polj med posameznimi okolji.

RAZVOJNO OKOLJE	IME KOMPONENTE	RAZRED V PROGRAMSKEM JEZIKU
Android studio	EditText	EditText
Xcode	TextField	UITextField
Xamarin studio	Entry	Entry

Slika 4.1: Prikaz osnovnih razlik komponente vnosno polje.

1. **Android studio (uporabljene lastnosti)**: layout_width, layout_height, id, Margin, background, hint, textColor, textColorHint, gravity in inputType.

   Preko layout_width in layout_height smo nastavili širino in višino vnosnega polja. Z lastnostjo id vnosno polje dobi unikatno ime. S pomočjo tega imena postane vnosno polje razvijalcu vidno v programski kodi (krmilnik). Lastnost Margin služi za odmike vnosnega polja. Z lastnostjo Background smo nastavili datoteko XML. V njej smo določili barvi ozadja in obrobe, debelino obrobe in stopinje robov obrobe. Datoteko XML se tako lahko uporabi na vseh željenih vnosnih poljih. Za namig uporabniku, katero informacijo je treba vnesti v posamezno vnosno polje, smo uporabili lastnost hint. TextColor in textColorHint nastavita barvo pisave v vnosnem polju. Za poravnavo besedila v vnosnem polju smo uporabili lastnost gravity. Nekatera vnosna polja so namenjena samo številkam ali črkam. Vrsto dovoljenega besedila nastavimo s pomočjo lastnosti inputType. Pri tem se prav tako določi vrsta gesla, ki vneseno besedilo pretvori v znak zvezdice (*).

2. **Xcode (uporabljene lastnosti)**: Width, Height, Color, Placeholder, Aligment, Margin, Background , Secure Text Entry in placeholderColor.

   Lastnosti Width in Height določata širino in višino vnosnega polja. Lastnost Color služi za določanje barve besedila. Za namig uporabnikom se uporablja lastnost Placeholder. Barva pisave namiga se določi s placeholderColor. Poravnavo pisave znotraj vnosnega

polja razvijalec določi z lastnostjo Aligment. Odmiki vnosnega polja se nastavijo s pomočjo lastnosti Margin. Barvo ozadja ali ozadje se določi na Background. Razvijalec ima tu možnost izbire poljubne slike. Za vnosno polje, ki služi kot geslo, mora razvijalec izbrati lastnost Secure Text Entry. Obrobe smo v programskem jeziku oblikovali znotraj krmilnika in ne na grafičnega urejevalnika. Unikatno ime vnosnega polja smo določili s pomočjo grafičnega urejevalnika vmesnika.

3. **Xamarin studio (uporabljene lastnosti)**: x:Name, TextColor, BackGroundColor, HorizontalTextAligment, Placeholder, Margin in IsPassword.

   Unikatno ime smo določili z lastnostjo x:Name. Barvo pisave izberemo s pomočjo lastnosti TextColor, barvo ozadja pa z BackGroundColor. Poravnava besedila v horizontalni/vertikalni smeri se določi s HorizontalTextAligment. Besedilo namiga prikaže okolje s pomočjo lastnosti Placeholder. Kot v prej omenjenih okoljih se tudi tu odmik nastavlja z lastnostjo Margin. Če želi razvijalec ustvariti vnosno polje za geslo, uporabi lastnost IsPassword.

4.1.2 Oznaka

Oznaka (angleško Label) je komponenta, namenjena prikazu posameznega tekstovnega podatka. Uporabniku aplikacije ne more neposredno spreminjati vsebine oznake. Komponenta oznaka primarno ni mišljena za interakcijo z uporabnikom, a jo razvijalec lahko uporabniku omogoči tudi v želji po interakciji. Primer bi bila povezava, namenjena navigaciji aplikacije, ki jo razvijalec mora podpreti in ni privzeto delovanje komponente oznaka. Osnovne razlike med okolji so vidne na spodnji sliki (Slika 4.2).

RAZVOJNO OKOLJE	IME KOMPONENTE	RAZRED V PROGRAMSKEM JEZIKU
Android studio	TextView	TextView
Xcode	Label	UILabel
Xamarin studio	Label	Label

Slika 4.2: Osnovne razlike komponente oznaka med okolji.

Nekatere lastnosti smo že spoznali in jih bomo na tem mestu samo omenili. Barva ozadja oznake je privzeto transparentna.

1. **Android studio (uporabljene lastnosti)**: text, Layout_Margin, layout_width, layout_height in textColor. Z lastnostjo text nastavimo vsebino besedila, ki bo prikazana uporabniku.

2. **Xcode (uporabljene lastnosti)** Text, Width, Height, Color in Margin. Lastnost Text tudi tu določi vsebino oznake, vidno uporabniku.

3. **Xamarin studio (uporabljene lastnosti)**: x:Name, Text, Margin in TextColor. Enako kot pri okolju Xcode tudi tu lastnost Text določi vsebino oznake.

4.1.3 Gumb

Gumb (angleško Button) je komponenta, preko katere lahko uporabnik v aplikaciji sproži določeno akcijo in spremeni potek izvajanja. Delovanje akcije je lahko poljubno in implementirati jo je potrebno na krmilniku pogleda. Akcijo nastavimo na Poslušalec Gumba (angleško event listnener). Za lažje razumevanje Poslušalca si ga lahko predstavljamo kot osebo, ki prenese sporočilo do krmilnika, in sicer, da je bil Gumb kliknjen in je potrebno izvesti določeno akcijo. Osnovne razlike razvojnih okolij so vidne na spodnji sliki (Slika 4.3).

RAZVOJNO OKOLJE	IME KOMPONENTE	RAZRED V PROGRAMSKEM JEZIKU
Android studio	Button	Button
Xcode	Button	UIButton
Xamarin studio	Button	Button

Slika 4.3: Osnovne razlike komponente Gumb med okolji.

Postopek določanja Poslušalca bomo prikazali kasneje. Določi se ga lahko na dva načina. Prvi je s pomočjo grafičnega urejevalnika lastnosti, drugi pa preko programske kode v krmilniku pogleda. V diplomski nalogi smo Poslušalce določali na drugi način. Gumb je pravzaprav oznaka z drugačnim privzetim videzom in namenom. Namenjena je interakciji z uporabnikom. Vse lastnosti oznake lahko uporabimo v komponenti Gumb, kar velja za vsa okolja.

1. **Android studio (uporabljene lastnosti)**: id, layout_width, layout_height, Layout_Margin, text, textColor in BackGround.

2. **Xcode (uporabljene lastnosti)**: Text, Width, Height, Color, Margin in Background.

3. **Xamarin studio (uporabljene lastnosti)**: x:Name, Text, Margin, BackGroundColor in TextColor.

4.1.4 Pogled slika

Pogled slika (angleško ImageView) je komponenta, namenjena prikazu poljubne slike. Osnovne razlike razvojnih okolji so vidne na sliki (Slika 4.4).

RAZVOJNO OKOLJE	IME KOMPONENTE	RAZRED V PROGRAMSKEM JEZIKU
Android studio	ImageView	ImageView
Xcode	Image View	UIImageView
Xamarin studio	Image	Image

Slika 4.4: Osnovne razlike komponente Pogled slika med okolji.

Vsebino Pogleda slike lahko nastavimo preko grafičnega urejevalnika v okoljih Android in Xcode. V Xamarin studiu je to mogoče programsko in v urejevalniku XAML. Programsko je mogoče nastaviti vsebino Pogleda slike tudi v ostalih dveh okoljih. Vse željene slike morajo biti dodane v projekt v ustrezne mape.

1. **Android studio (uporabljene lastnosti)**: id, layout_width, layout_height, Layout_Margin in BackGround.

2. **Xcode (uporabljene lastnosti)**: Width, Height in Margin. Posebnost okolja Xcode je prikaz vseh slik v grafičnem urejevalniku vmesnika. Razvijalec lahko povleče željeno sliko v komponento in ta se nastavi. Nabor slik se nanaša na vsebino mape s slikami v projektu.

3. **Xamarin studio (uporabljene lastnosti)**: x:Name, Margin in Source. Lastnost Source določa sliko, ki bo prikazana.

4.1.5 Pogled seznam

Pogled seznam (angleško ListView) je komponenta, ki prikazuje poljubne podatke v obliki seznama. Sestavljen je iz celic, ki se lahko oblikujemo po potrebi. Celice imajo svoje Poslušalce, preko katerih lahko uporabnik izvede določeno akcijo. Primer uporabe komponente Seznam so sporočila na mobilnih napravah. V seznamu sporočil lahko vidimo kratek opis sporočila in pošiljatelja. S klikom na določeno sporočilo (celica) se odpre okno, v katerem lahko izbranemu pošiljatelju napišemo odgovor. Osnovne razlike Pogleda seznam med okolji so vidne na spodnji sliki (Slika 4.5).

RAZVOJNO OKOLJE	IME KOMPONENTE	RAZRED V PROGRAMSKEM JEZIKU
Android studio	ListView	ListView
Xcode	Table View	UITableView
Xamarin studio	ListView	ListView

Slika 4.5: Osnovne razlike komponente Pogled seznam med okolji.

Vsebina in število celic se nastavi pred prikazom grafičnega vmesnika aplikacije na krmilniku posameznega pogleda. Implementacija bo prikazana kasneje v poglavju.

1. **Android studio (uporabljene lastnosti)**: id, layout_width, layout_height in Layout_Margin.

2. **Xcode (uporabljene lastnosti)**: Width, Height, Margin in Background.

3. **Xamarin studio (uporabljene lastnosti)**: x:Name, Margin in BackGroundColor.

4.1.6 Pogled

Pogled (angleško View) je po hierarhiji nadrejen ostalim komponentam. Služi kot platno, na katerega lahko razvijalec postavi ostale komponente. Je obvezna komponenta vsakega grafičnega vmesnika. Komponente lahko gnezdimo poljubno. Pogled je okno aplikacije. Osnovne razlike med okolji so vidne na spodnji sliki (Slika 4.6).

RAZVOJNO OKOLJE	IME KOMPONENTE	RAZRED V PROGRAMSKEM JEZIKU
Android studio	Layout	Layout
Xcode	View Controller	UIViewController
Xamarin studio	ContentPage	ContentPage

Slika 4.6: Osnovne razlike komponente Pogled med okolji.

Ob ustvarjanju novega okna aplikacije znotraj vseh treh okolij te privzeto vsebujejo prazno komponento Pogled. Njena velikost je enaka velikosti izbrane naprave. Barva ozadja je privzeto bela. Razvijalec lahko tako nenadoma začne z oblikovanjem videza aplikacije. Pogledi vsebujejo pravila ali orientacijo za razporeditev komponent, ki jih razvijalec postavi in umesti na Pogled. Spodaj bomo omenili samo uporabljena razvrščanja.

1. **Android studio (uporabljene lastnosti)**: id, layout_width, layout_height, Layout_Margin in background.

   V okolju Andorid studio smo za razvoj aplikacije uporabili LinearLayout in RelativeLayout. LinearLayout komponente razporedi v vrsto (od zgoraj navzdol ali z leve proti desni). RelativeLayout omogoča poljubno razvrščanje komponent v Pogledu. Razvrščanja v Android studiu hkrati predstavljajo tudi okno aplikacije.

2. **Xcode (uporabljene lastnosti)**: Width, Height, Margin in Background. Komponenta ViewController predstavlja eno okno v aplikaciji. Razvrščanje komponent je poljubno.

3. **Xamarin studio (uporabljene lastnosti)**: ContentPage predstavlja okno aplikacije. Znotraj okna mora razvijalec sam določiti razvrščanje. Trenutno se v Xamarin.Forms najpogosteje uporablja razvrščanje v mrežo. Razvijalec vnaprej opredeli število vrstic in stolpcev ter s koordinatami komponento postavi na željeno mesto. To razvrščanje je za večplatformski razvoj najboljše, saj so odstopanja pri postavitvi komponent med platformami zelo majhna v primerjavi z ostalimi.

4.1.7 Izbirnik

Komponenta Izbirnik (angleško Picker) razvijalcem omogoča omejitev uporabnika na izbiro določenih podatkov. Vrednosti znotraj Izbirnika določi razvijalec programsko. Privzeto je izbrana prva možnost iz nabora vrednosti. Ko uporabnik izbere Izbirnik, se pojavi spustni seznam z naborom vrednosti. S klikom na določeno vrednost se nanjo nastavi določena vrednost uporabnika in spustni seznam se zapre. Grafični prikaz Izbirnika se med platformami razlikuje, vendar konceptualno v vseh deluje enako. Osnovne razlike med komponento Izbirnik v različnih razvojnih okoljih so vidne na spodnji sliki (Slika 4.7).

RAZVOJNO OKOLJE	IME KOMPONENTE	RAZRED V PROGRAMSKEM JEZIKU
Android studio	Spinner	Spinner
Xcode	Picker view	UIPickerView
Xamarin studio	Picker	Picker

Slika 4.7: Osnovne razlike komponente Izbirnik med okolji.

Za beleženje sprememb znotraj komponente služijo Poslušalci. Z njihovo pomočjo lahko razvijalci na izbrano vrednost implementirajo potrebno logiko aplikacije.

1. **Android studio (uporabljene lastnosti)**: id, layout_width, layout_height, Layout_Margin in background. Z metodo setAdapter komponenti nastavimo seznam vrednosti. Posebnost okolja Android studio je definiranje teh vrednosti v datoteki XML (string-array) in nato sklicevanje na ta seznam.

2. **Xcode (uporabljene lastnosti)**: Width, Height in Background. Ustvarili smo novi krmilnik in pogled, ki je dedoval lastnosti UIPickerView razreda. Preko teh lastnosti smo določili nabor vrednosti seznama in Poslušalce komponente Izbirnik na pogledu.

3. **Xamarin studio (uporabljene lastnosti)**: x:Name, Margin,TextColor in BackGroundColor. Razred Picker ima v programskem jeziku C# svojo seznam, ki

je po privzetem prazen. Za prikaz neke vrednosti je treba željeno vrednost dodati v seznam komponente.

4.1.8 Izbirnik datuma

Komponenta Izbirnik datuma ponuja uporabniku izbiro datuma. V osnovni deluje enako kot Izbirnik, le da so podatki omejeni na tip datuma. Ker aplikacijo lahko uporabljajo posamezniki z različnih koncev sveta, komponenta razvijalcu reši problem lokalnega prikazovanja datuma (nastavljeno glede na nastavitve mobilne naprave).

Komponenta poskrbi za poenoten format in ustreznost datuma. Osnovne razlike s komponento Izbirnik datuma so vidne na spodnji sliki (Slika 4.8).

RAZVOJNO OKOLJE	IME KOMPONENTE	RAZRED V PROGRAMSKEM JEZIKU
Android studio	DatePickerDialog	DatePickerDialog
Xcode	Date Picker	UIDatePicker
Xamarin studio	DatePicker	DatePicker

Slika 4.8: Osnovne razlike komponente Izbirnik datuma med okolji.

1. **Android studio (uporabljene lastnosti)**: V tem okolju smo implementirali Izbirnik datuma s pomočjo komponente Gumb. Ob njegovi izbiri smo prikazali DatePickerDialog. Grafično te komponente nismo spreminjali in pustili smo privzete nastavitve. Potrebna je bila implementacija Poslušalca.

2. **Xcode (uporabljene lastnosti)**: Width in Height. Datum je privzeto nastavljen na trenutni datum naprave. Poslušalec beleži vsako spremembo izbire datuma in razvijalec ga ne rabi implementirati. Do trenutne izbire uporabnika pridemo s pomočjo metode Date.

3. **Xamarin studio (uporabljene lastnosti)**: x:Name, Margin, BackGroundColor in Format. Z lastnostjo Format razvijalec nastavi način prikaza datuma (primer »dd.MM.yyyy«). Tako kot pri okolju Xcode razvijalcu Poslušalca ni potrebno implementirati. Do izbranega datuma pridemo z metodo Date. Privzeto je izbran trenutni datum naprave.

4.1.9 Stikalo

Komponenta Stikalo (angleško Switch) nudi uporabniku dve izbiri. Ob kliku na komponento se stanje zamenja in vedno preide v drugo izbiro. V ciljni aplikaciji diplomske naloge smo Stikalo uporabili za določanje, ali gre za javni prevoz ali ne (potni strošek). Stikalo je smiselno uporabiti, ko od uporabnika pričakujemo izbiro »da« ali »ne«. Osnovne razlike med komponento Izbirnik datuma med različnimi razvojnimi okolji so vidne na spodnji sliki (Slika 4.9).

RAZVOJNO OKOLJE	IME KOMPONENTE	RAZRED V PROGRAMSKEM JEZIKU
Android studio	Switch	Switch
Xcode	Switch	UISwitch
Xamarin studio	Switch	Switch

Slika 4.9: Osnovne razlike komponente Stikalo med okolji.

Stikalo se zaveda svojega stanja (prvo ali drugo). Razvijalec tako lahko implementira ustrezno logiko glede na stanje Stikala preko Poslušalcev.

1. **Andorid studio (uporabljene lastnosti)**: Pri razvoju domorodne aplikacije Android zaradi težav z videzom Stikala na grafičnem vmesniku nismo uporabili te komponente. Izvor težave je neznan in se pojavlja na osebnem računalniku diplomanta. Velikost komponente je bila premajhna in ni bilo razvidno, da gre za stikalo. Za rešitev smo uporabili komponento Izbirni gumb (**Error! Reference source not found.**).

2. **Xcode (uporabljene lastnosti)**: Width, Height in Margin.

3. **Xamarin studio (uporabljene lastnosti)**: x:Name in VerticalOptions. Z lastnostjo VerticalOptions določimo poravnavo komponente.

4.1.10 Izbirni gumbi

Komponenta Izbirni gumb (angleško Radio button) deluje podobno kot Izbirnik. Nudi nabor določenih vrednosti, med katerimi lahko uporabnik izbira. Vrednosti je manj kot v seznamih in smiselno jih je prikazati druge ob drugih brez spustnega seznama. V aplikaciji Android smo tako prikazali izbiro »da« ali »ne« za javni prevoz (potni strošek). To komponento smo uporabili izključno zaradi težav s komponento Stikalo, omenjeno v poglavju **Error! Reference source not found.**. Izbirni gumbi v okolju Xamarin studio in Xcode niso podprti.

1. **Androdi studio (uporabljene lastnosti)**: id, layout_width, layout_height, Layout_Margin, text in textColor. Na grafični vmesnik aplikacije lahko postavimo poljubno število Izbirnih gumbov. Vsak predstavlja eno izbiro. Po privzetem Izbirni gumb ni izbran. Preko metode isCheked razvijalec dobi stanje Izbirnega gumba. S pomočjo Poslušalcev lahko implementiramo ustrezno logiko ob spreminjanju stanja.

4.1.11 Ugotovitve

V večini primerov imajo komponente ustrezne preslikave (obstajajo) znotraj drugih platform. Komponente pogosto delijo podobna imena. Komponente s povsem drugačnimi imeni lahko za neizkušenega razvijalca pomenijo časovno potrato pri iskanju komponente in so moteč dejavnik. Pri razvoju obeh platform v domorodnih okoljih Android studio in Xcode je bilo potrebno paziti na izbiro komponent na grafičnem vmesniku (enotnost). Xamarin studio je to skrb razvijalcu odvzel, saj ogrodje Xamarin.Forms podpira le komponente, ki imajo preslikavo v ciljnih domorodnih platformah (Android in iOS). Enotnost imen in preslikav v ustrezne komponente je še ena prednost razvojnega okolja Xamarin studio. Vse uporabljene lastnosti pri razvoju ciljne aplikacije so podprte v vseh treh okoljih, razlike so le imenske. Nobeno okolje pri primerjavi (lastnosti komponent) nima prednosti, kar velja za uporabljene lastnosti pri razvoju ciljne aplikacije diplomske naloge.

4.2 Primerjava končnega videza grafičnega vmesnika aplikacije med domorodnima aplikacijama in aplikacijo Xamarin.Forms

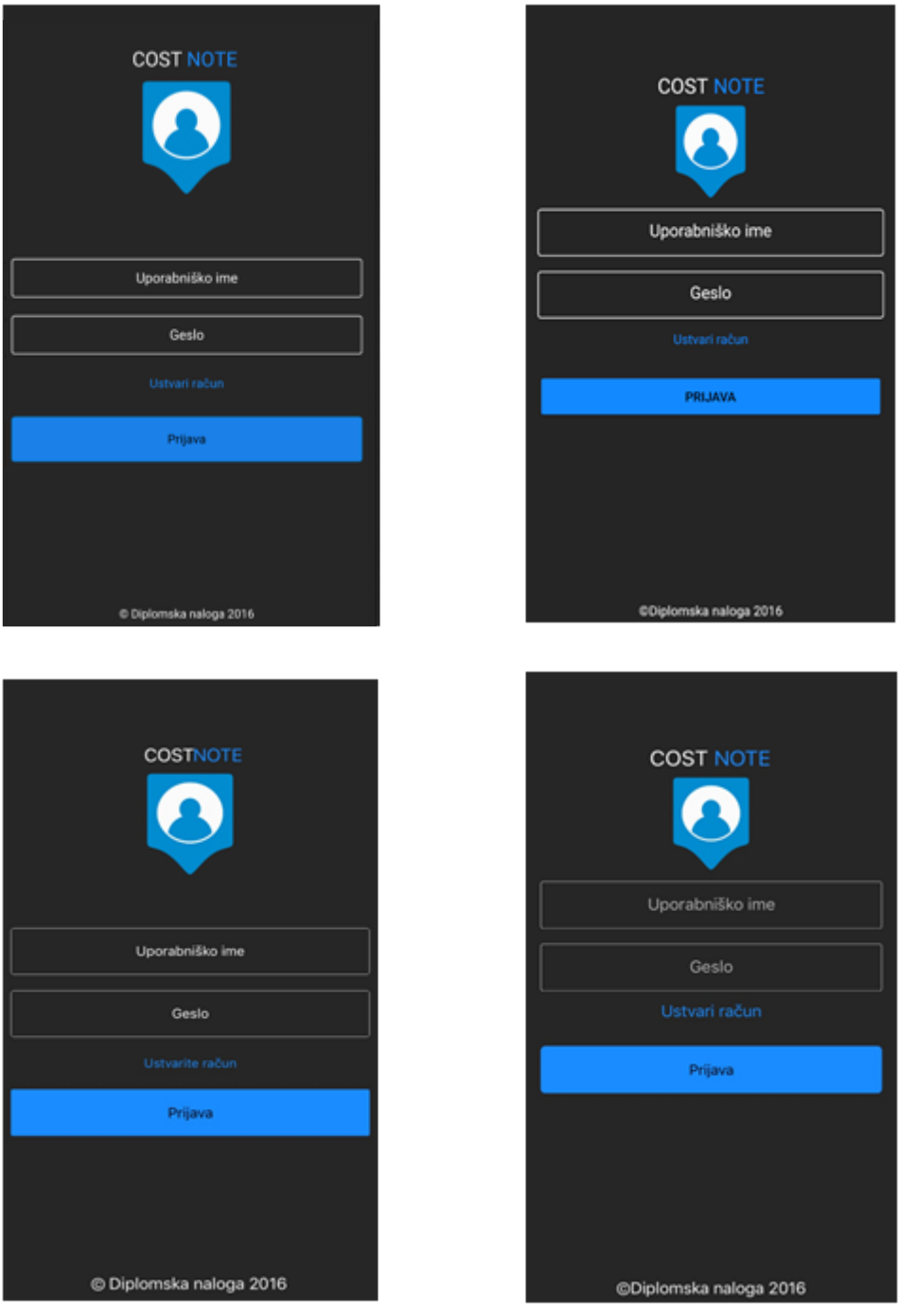
V tem poglavju bomo predstavili končni videz vseh treh aplikacij. Opaziti bo mogoče manjše razlike, ki jih bomo prikazali na priloženih slikah. Razvoj aplikacij je potekal v naslednjem vrstnem redu:

1. Domorodna aplikacija Android (Android studio).
2. Domorodna aplikacija iOS (Xcode).
3. Večplatformski razvoj v Xamarin.Forms (Xamarin studio).

Vrstni red razvoja je bil pomemben za končni videz aplikacije, ki je bil zasnovan pri razvoju prve aplikacije v Android studiu. Cilj je bil videz prve aplikacije poustvariti v ostalih okoljih. Aplikacija je morala na vseh platformah delovati enako in podpreti vse funkcionalne zahteve, zastavljene v poglavju 1.1.

4.2.1 Prikaz videza vseh oken aplikacije

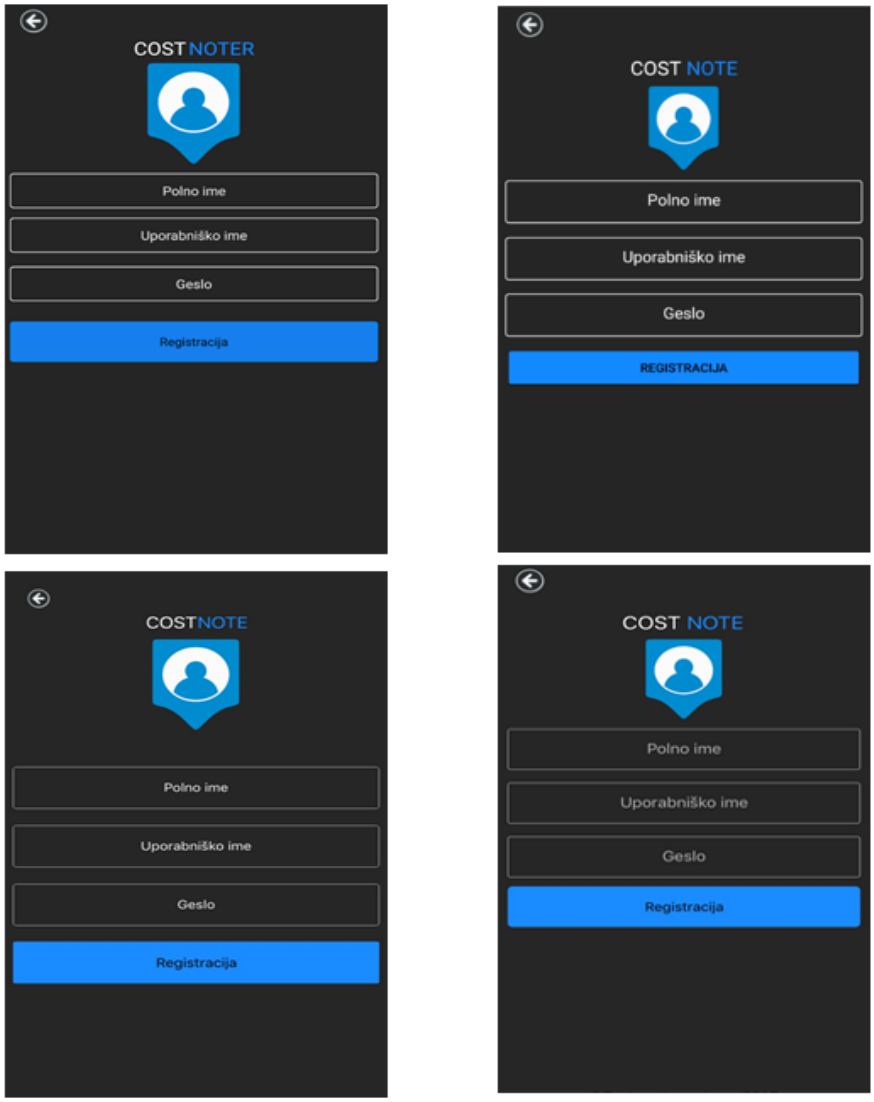
Okno za vpis uporabnika (Slika 4.10) se prikaže kot zaženo okno aplikacije.



Slika 4.10: Prikaz okna Vpis uporabnika. Levo zgoraj (Android), levo spodaj (iOS), desno zgoraj (Xamarin.Android) in desno spodaj (Xamarin.iOS).

Barve so med aplikacijami usklajene. Določene so bile v fazi razvoja prve aplikacije in so služile kot zgled ostalim. Razlike so vidne v postavitvi komponent. Zaradi uporabe mrežaste razporeditve na platformi Xamarin.Forms videza obeh domorodnih aplikacij ni bilo mogoče povsem natančno poenotiti. Posledica je postavitev vnosnih polj višje na oknu. Velikosti nekaterih elementov so drugačne tudi v primerjavi z domorodnima aplikacijama. Iz neznanega razloga je tudi pisava pri gumbih aplikacije Xamarin.Android z veliki črkami (Xamarin.iOS nima te težave).

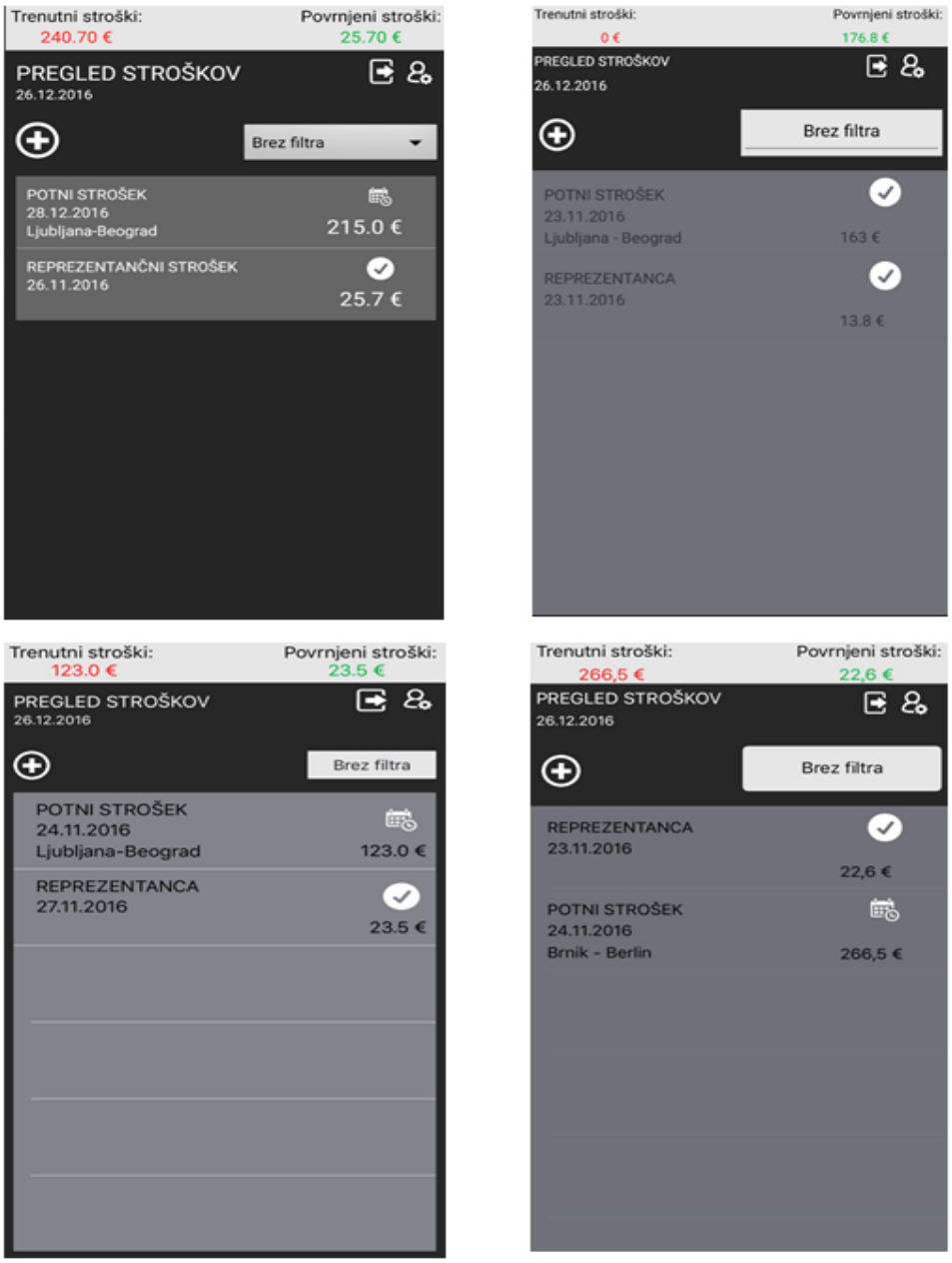
Ob izbiri ustvarjanja računa se odpre okno registracije uporabnika (Slika 4.11).



Slika 4.11: Prikaz okna Registracija uporabnika. Levo zgoraj (Android), levo spodaj (iOS), desno zgoraj (Xamarin.Android) in desno spodaj (Xamarin.iOS).

Vidne so nekonsistentne poravnave in odmiki elementov. Velikost Gumba v aplikaciji Xamarin.Android je nekoliko manjša od gumba v domorodni aplikaciji Android.

Ob uspešnem vpisu v aplikacijo se odpre osrednje okno aplikacije (Slika 4.12). To okno uporabniku služi za dostop do vseh funkcionalnosti aplikacije.

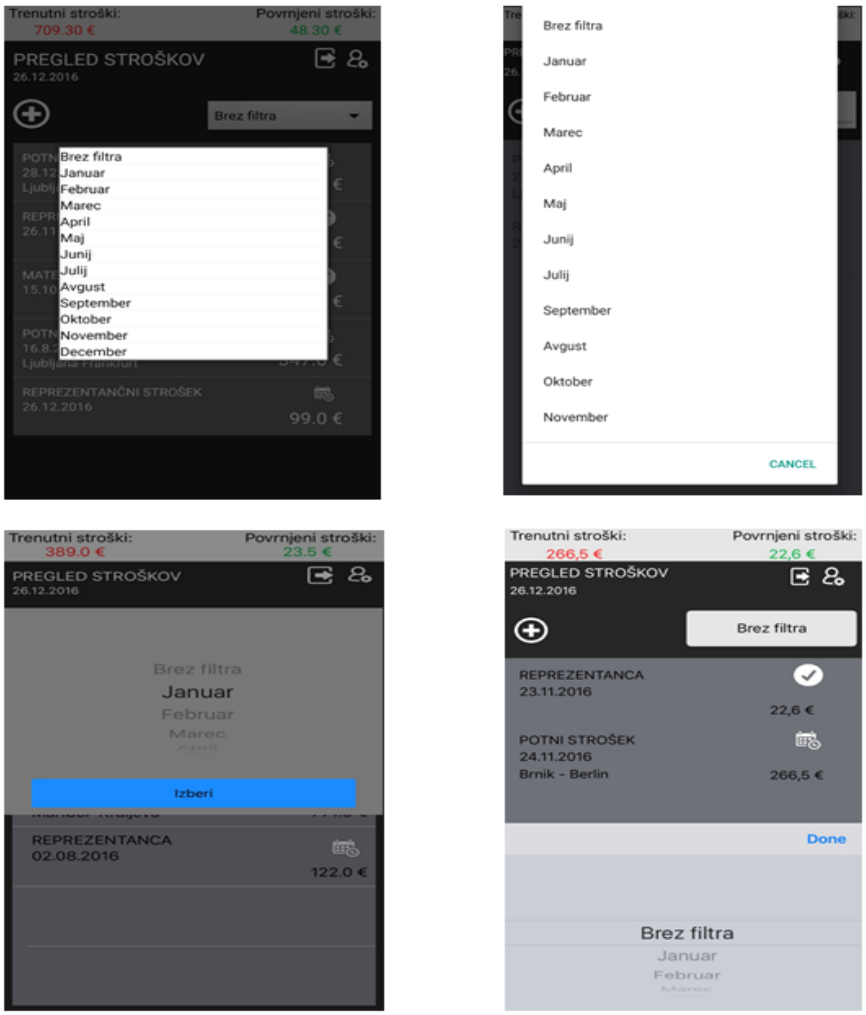


Slika 4.12: Prikaz osrednjega okna. Levo zgoraj (Android), levo spodaj (iOS), desno zgoraj (Xamarin.Android) in desno spodaj (Xamarin.iOS).

Okna se v osnovi ne razlikujejo, je pa vidna prva manjša pomanjkljivost večplatformskega razvoja. Komponenta Pogled seznam (poglavje 4.1.5) ne vsebuje skupne lastnosti odmika v

platformi Xamarin.Forms. Odmika od robov pri aplikacijah Xamarin.iOS in Xamarin.Android v primerjavi z domorodnima aplikacijama ni. Odmik se lahko implementira v podprojektih določene platforme s pomočjo »custom renderejev«. Njihova implementacija bo prikazana kasneje v poglavju. Razlog za neimplementacijo odmika pri komponenti Pogled seznam je preveč vloženega dela (kompleksnost) za doseganje samo ene lastnosti. Če bi potrebovali še kakšno lastnost več, bi »custom renderer« implementirali po potrebi.

Ob izbiri prikaza stroškov za določen mesec se odpre spustni seznam z možnimi izbirami (Slika 4.13).

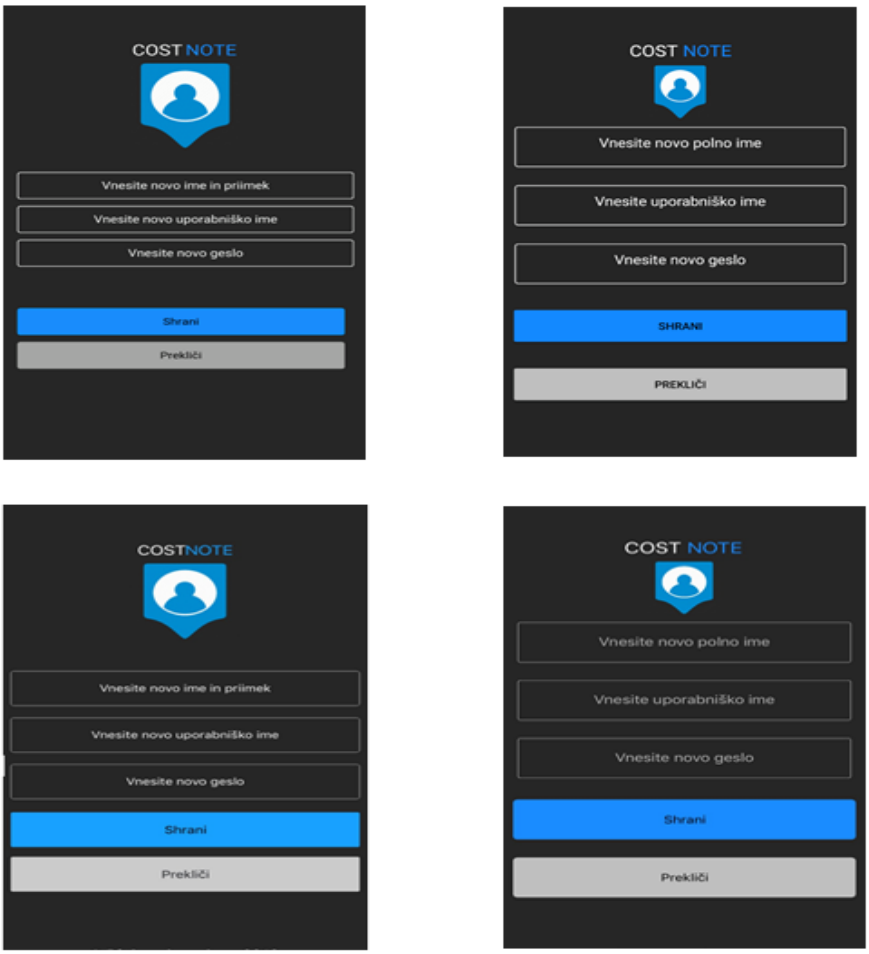


Slika 4.13: Prikaz izbrane komponente Izbirnik. Levo zgoraj (Android), levo spodaj (iOS), desno zgoraj (Xamarin.Android) in desno spodaj (Xamarin.iOS).

Razlike v videzu pojavnih oken izbire komponente Izbirnik so hitro opazne. V domorodnih aplikacijah privzete teme ciljnih mobilnih platform (Andorid 23 in iOS 10.0.2) nismo uporabljali. V domorodni aplikaciji Android smo uporabili zgolj spustni seznam brez dodatnih

popravkov videza. Aplikacija iOS je zahtevala razvoj pojavnega okna in tega ob izbiri komponente Izbirnik ne podpira privzeto. Videz pojavnega okna smo skušali barvno poenotiti z aplikacijo. V pojavno okno smo nato postavili komponento Izbirnik. Druga rešitev bi bila sistemsko okno, ki se pojavi v spodnjem delu aplikacije s komponento Izbirnik. Ker je razvoj potekal zaporedno in smo želeli domorodno aplikacijo iOS poenotiti z domorodno aplikacijo Android, smo izbrali prvo implementacijo pojavnega okna. Aplikaciji Xamarin.Forms uporabljata privzeto temo ciljnih mobilnih aplikacij (Andorid 23 in iOS 10.0.2). Videz Android pojavnih oken se razlikuje. Xamarin.iOS aplikacija je pojavno okno implementirala s pomočjo sistemskega pojavnega okna.

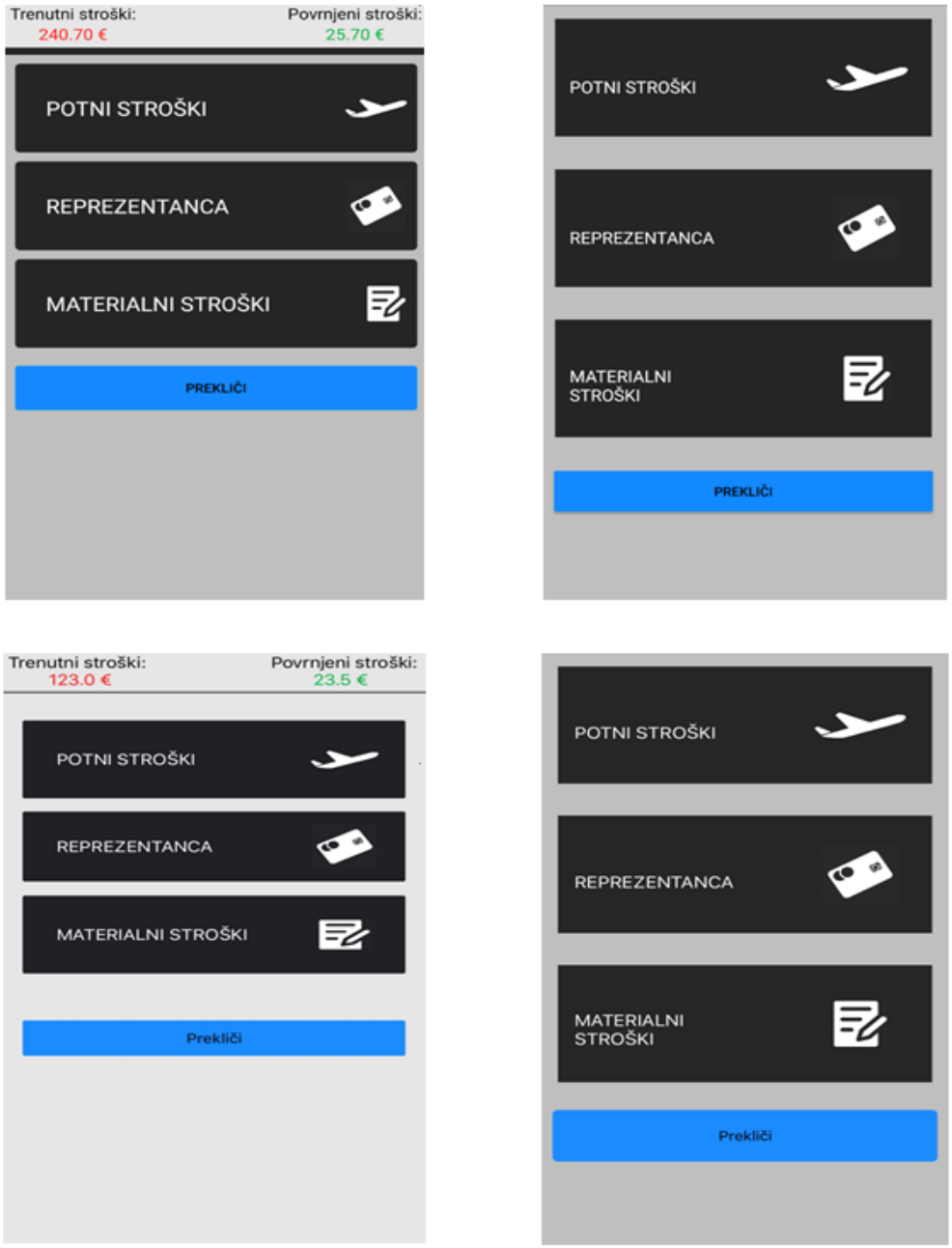
Ob izbiri ikone za urejanje uporabnika nas aplikacija preusmeri na ustrezno okno (Slika 4.14).



Slika 4.14: Prikaz okna za urejanje uporabnika. Levo zgoraj (Android), levo spodaj (iOS), desno zgoraj (Xamarin.Android) in desno spodaj (Xamarin.iOS).

Okno Urejanje uporabnika ima enake lastnosti kot okno vpisa in registracije uporabnika. Vidni so različni zamiki in velikosti komponent.

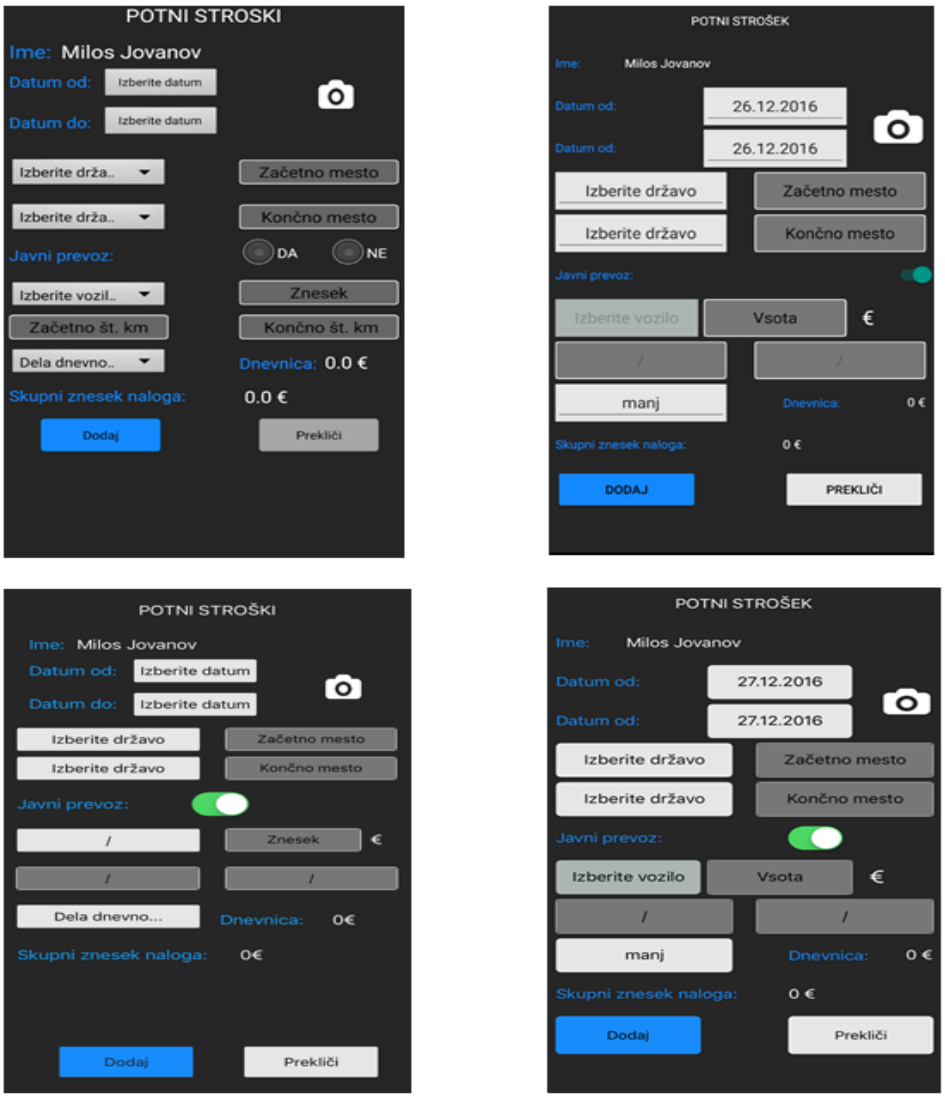
Ob izbiri ikone za dodajanje stroška se v domorodnih aplikacijah odpre pojavno okno z možnimi izbirami. Platforma Xamarin.Forms ne podpira razvoja pojavnih oken. Izbira stroškov se prikaže v novem oknu aplikacije. Videz okna za izbiro stroška je viden na spodnji sliki (Slika 4.15).



Slika 4.15: Prikaz okna za izbiro stroška. Levo zgoraj (Android), levo spodaj (iOS), desno zgoraj (Xamarin.Android) in desno spodaj (Xamarin.iOS).

Razlik med videzom aplikacij Xamarin.Forms skoraj ni (velikost gumba). Domorodna aplikacija iOS ima nekaj odtenkov svetlejšo barvo ozadja (izbrana barva je enaka v vseh aplikacijah). Zgornja vrstica s stanjem stroškov v aplikacijah Xamarin.Forms ni vidna (okna niso pojavna).

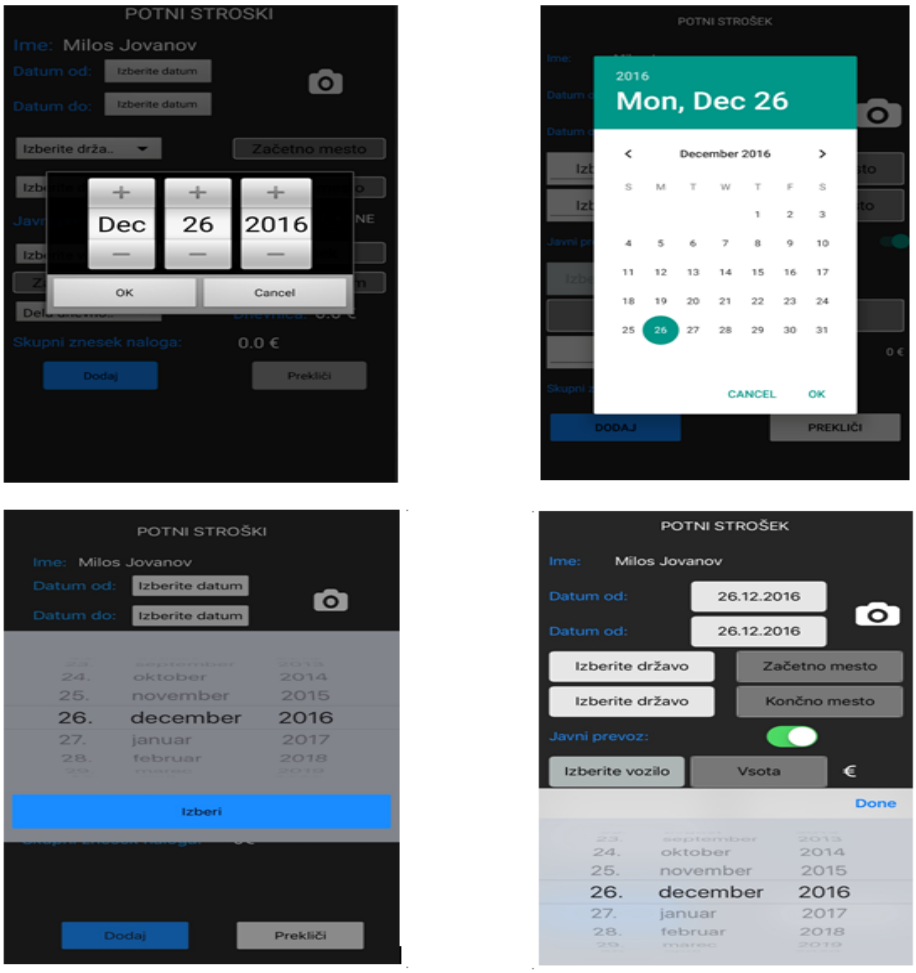
Ob izbiri potnega stroška se pojavi novo okno za vnos ustreznih podatkov (Slika 4.16).



Slika 4.16: Prikaz okna za dodajanje potnega stroška. Levo zgoraj (Android), levo spodaj (iOS), desno zgoraj (Xamarin.Android) in desno spodaj (Xamarin.iOS).

Bistvena razlika v oknu domorodne aplikacije Android je neuporaba komponente Stikalo zaradi že omenjenih težav. Domorodna aplikacija Android uporablja komponento Izbirni gumbi, ki pri ostalih aplikacijah ni prisotna.

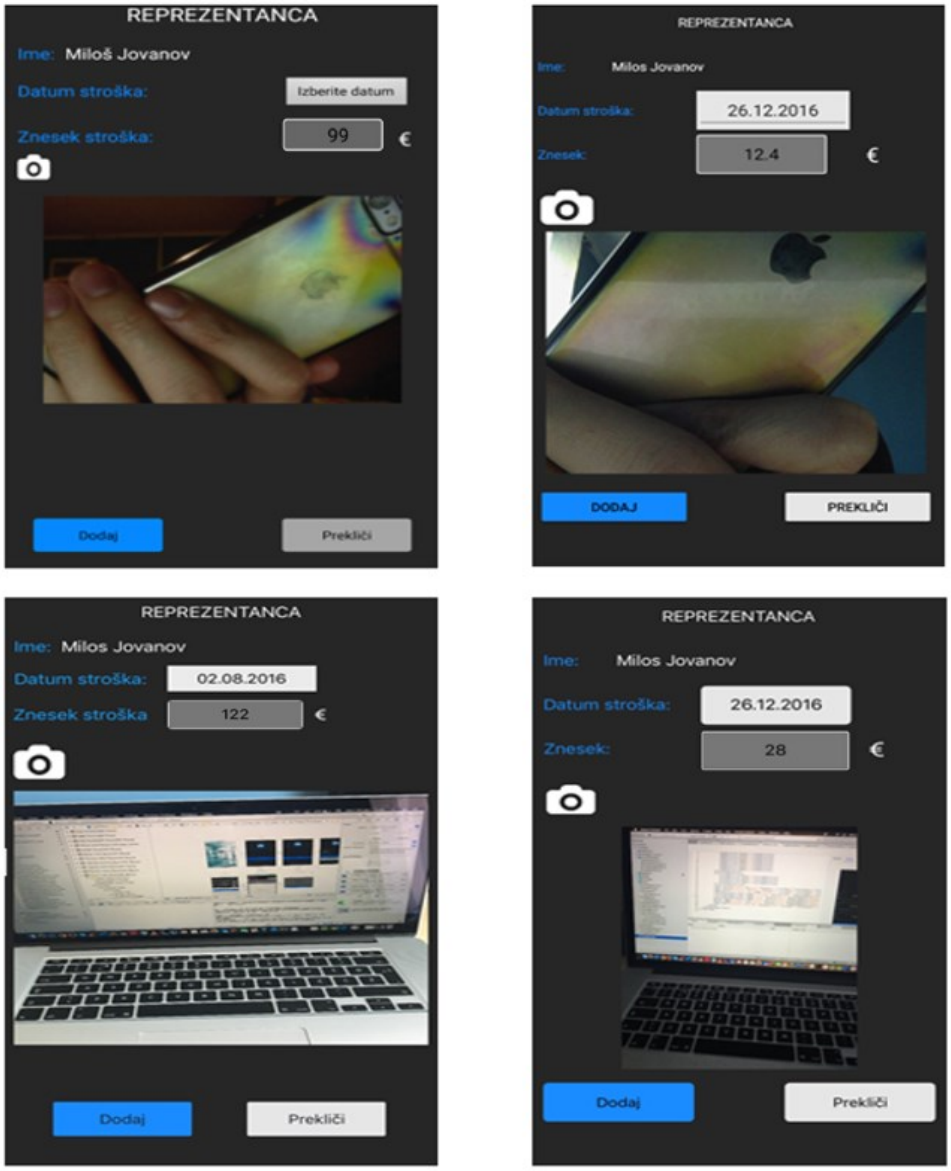
Vidne so razlike v postavitvi komponent in velikosti. Najbolj opazna je nekonsistentna poravnava komponente Stikalo v aplikacijah Xamarin.Forms. Stikalo v aplikaciji Xamarin.iOS je poravnano podobno kot pri domorodni aplikaciji iOS. Iz neznanih razlogov je poravnava Stikala v Xamarin.Android postavljena na desno stran okna, čeprav uporabljata enako poravnavo in postavitev v mreži (razporeditev okna v deljenem projektu) kot aplikacija Xamarin.iOS. Komponenta Izbirniki datuma v aplikacijah Xamarin.Forms ima prevzeto nastavljen datum mobilne naprave. Ob izbiri datuma se prikaže pojavno okno. Xamarin.Android prikaže pojavno okno v obliko koledarja, iz katerega uporabnik nato izbere datum. Koledar je prikazan zaradi privzete teme Android verzije (Android 23). V aplikaciji Xamarin.iOS se pojavi sistemsko pojavno okno z Izbirnikom datuma. V domorodni aplikaciji iOS smo Izbirnik datuma postavili v razvito pojavno okno. To komponento smo v domorodni aplikaciji Android prikazali s podprtim pojavnim oknom, ki se razlikuje od koledarskega prikaza. Pojavna okna za izbiro datuma so vidna na spodnji sliki (Slika 4.17).



Slika 4.17: Prikaz komponente Izbirnik datuma. Levo zgoraj (Android), levo spodaj (iOS), desno zgoraj (Xamarin.Android) in desno spodaj (Xamarin.iOS).

Okna za izbiro datuma so povsem nekonsistentna. Prvi razlog je sistemska razlika med mobilnimi platformami (prikaz komponent), drugi pa zaporedni razvoj aplikacij in razvoj domorodnih aplikacij brez izbrane teme, ki je specifična za določeno platformo (razvoj po meri in občutku razvijalca).

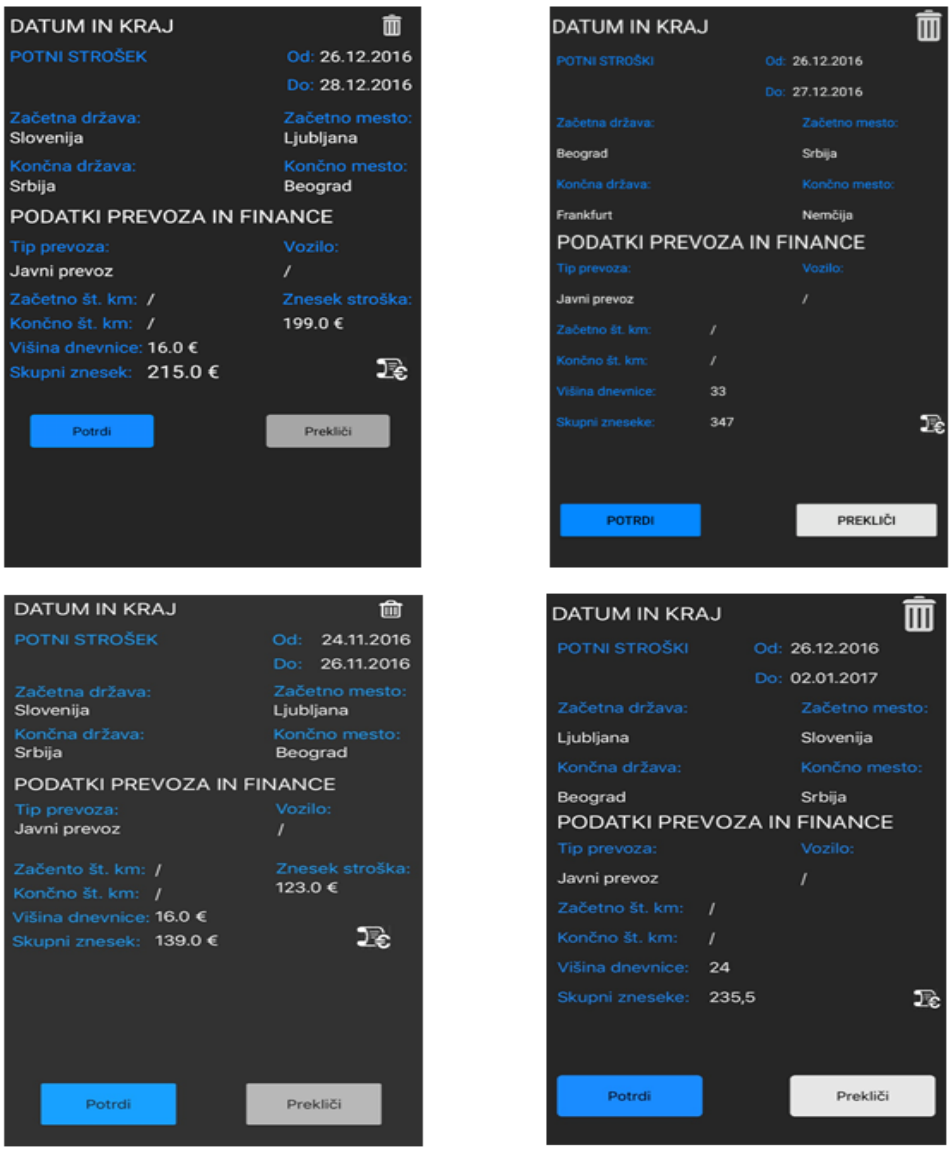
V oknu za izbiro vrste stroška (Slika 4.15) lahko uporabnik poleg potnega stroška izbere tudi materiali ali reprezentančni strošek. Z izbiro materialnega ali reprezentančnega stroška aplikacija odpre novo okno (Slika 4.18). Okno je za oba stroška enako (razlika je samo v naslovu).



Slika 4.18: Prikaz okna za dodajanje reprezentančnega ali materialnega stroška. Levo zgoraj (Android), levo spodaj (iOS), desno zgoraj (Xamarin.Android) in desno spodaj (Xamarin.iOS).

Vidna razlika je v tem oknu velikost komponente Pogled slika. V domorodni aplikaciji Android zaradi težav z razvrščanjem komponent znotraj okna velikosti ni bilo mogoče povečati. Težave so bile spremembe postavitve drugih komponent znotraj glavnega okna. V aplikacijah Xamarin.Forms je prisotna razlika med velikostjo komponente Vnosno polje. Velikost komponente Pogled slika se med aplikacijama Xamarin.Forms prav tako opazno razlikuje. Razlike so posledica razvrščanja komponent znotraj okna (mreža).

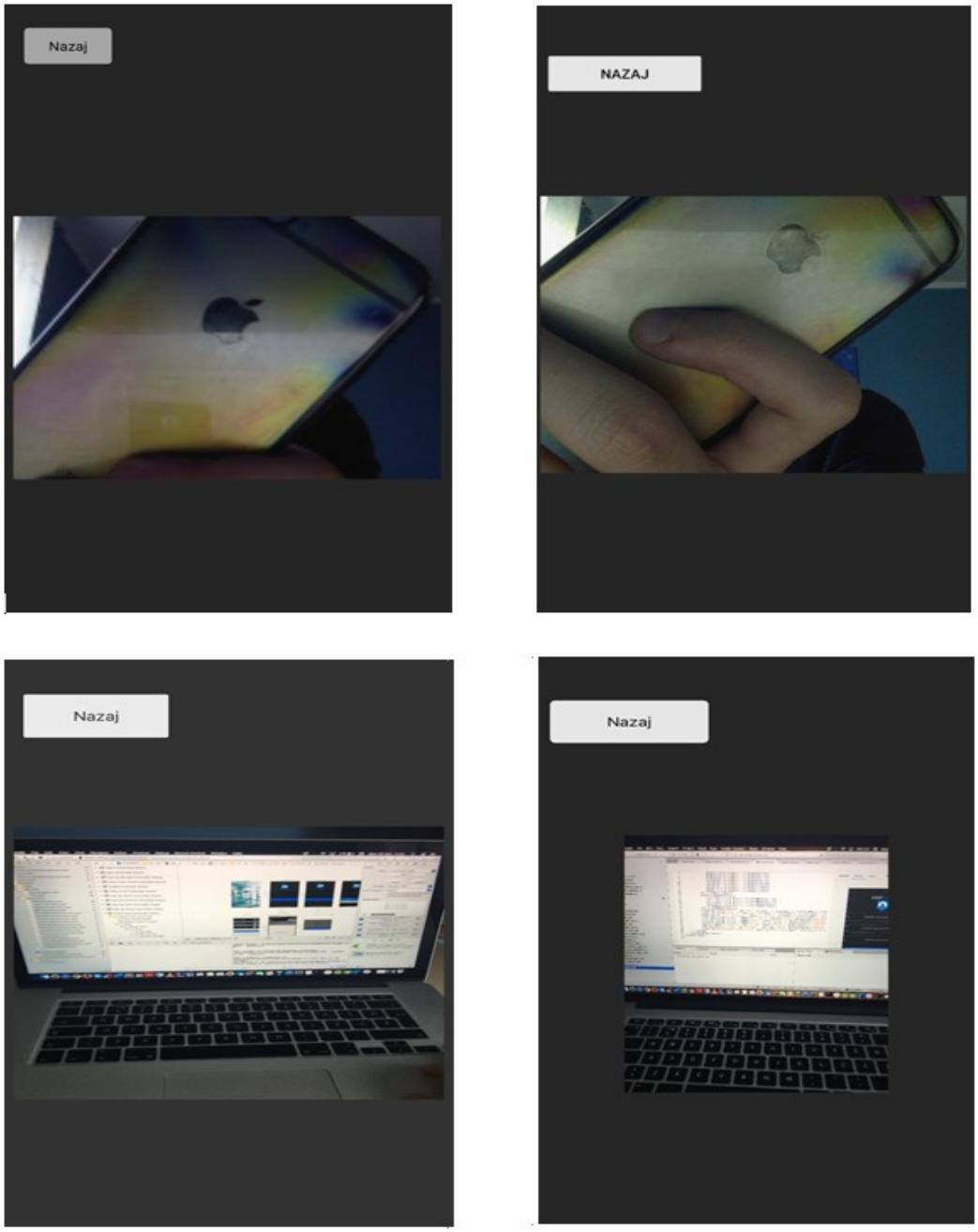
Ob izbiri že obstoječega stroška na seznamu osrednjega okna aplikacije (Slika 4.12) se odpre okno Podrobnosti stroška. Ob izbiri potnega stroška se odpre okno, prikazano na spodnji sliki (Slika 4.19).



Slika 4.19: Prikaz okna podrobnosti potnega stroška. Levo zgoraj (Android), levo spodaj (iOS), desno zgoraj (Xamarin.Android) in desno spodaj (Xamarin.iOS).

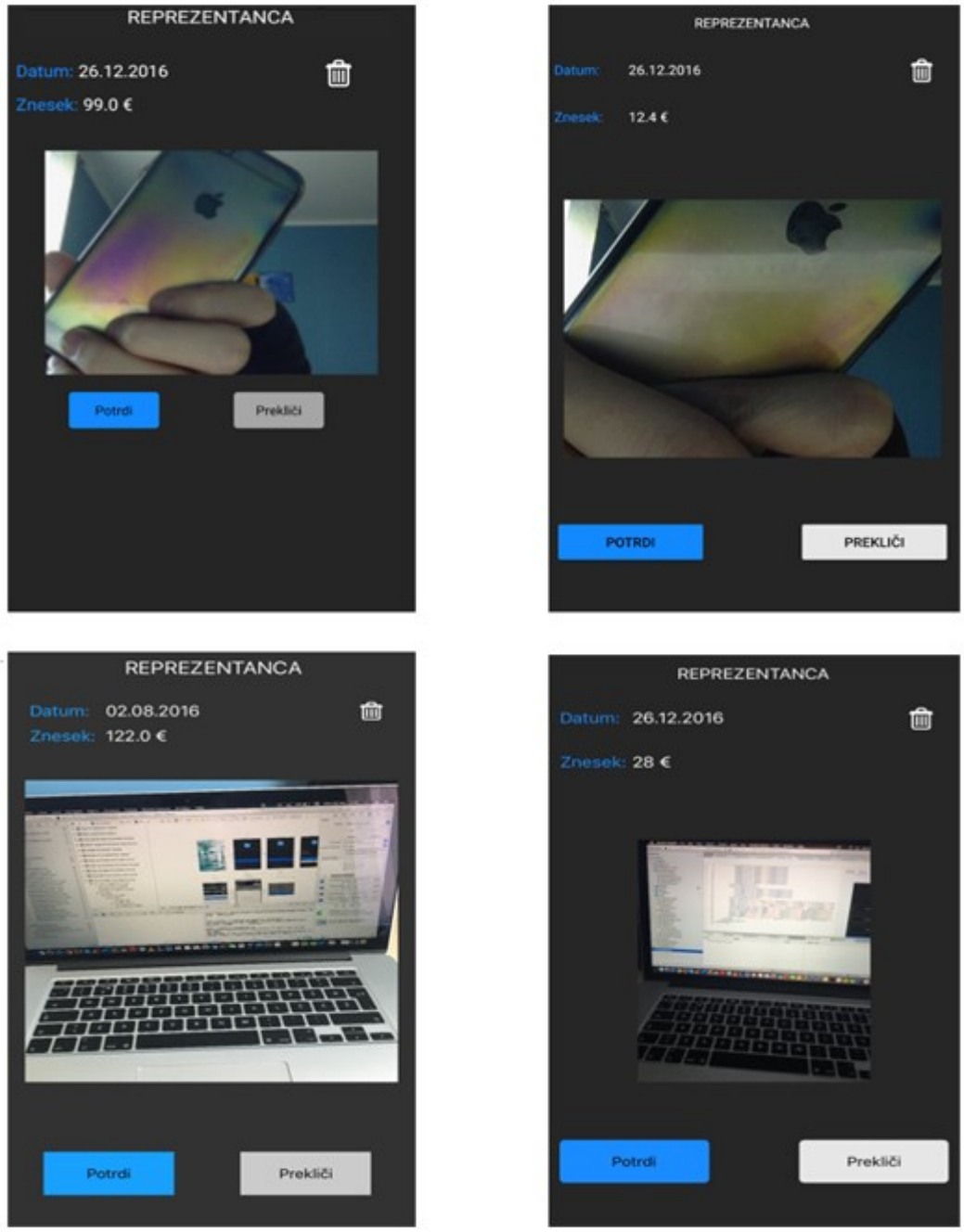
Domorodna aplikacija iOS ima vidno svetlejšo barvo ozadja, klub temu da je izbrana barva enaka vsem ostalim. Opazne so razlike v poravnavah in postavitvah komponent ter velikostih slik (ikone).

Ob kliku na ikono račun se odpre okno za prikaz slike (Slika 4.20).



Slika 4.20: Okno za prikaz slike potnega stroška. Levo zgoraj (Android), levo spodaj (iOS), desno zgoraj (Xamarin.Android) in desno spodaj (Xamarin.iOS).

Ob izbiri materialnega ali reprezentančnega stroška aplikacija odpre okno za prikaz podrobnosti (Slika 4.21).



Slika 4.21: Prikaz okna podrobnosti materialnega ali reprezentančnega stroška. Levo zgoraj (Android), levo spodaj (iOS), desno zgoraj (Xamarin.Android) in desno spodaj (Xamarin.iOS).

4.2.2 Ugotovitve

Na podlagi prikazanega grafičnega vmesnika lahko rečemo, da sta aplikaciji, razviti na platformi Xamarin.Forms, bolj podobni ena drugi kot domorodni aplikaciji Android in iOS. Pomanjkljivost aplikacij Xamarin.Forms je različna postavitev, poravnava in velikost komponent na oknu aplikacije (kljub enaki postavitvi, poravnavi in velikosti v skupnem projektu). Pri razvoju domorodne aplikacije Android (Android studio) smo imeli težave z razvrščanjem komponent. Občasna odstopanja postavitve gumbov so posledica te težave, ki bi jih lahko odpravili z izbiro drugačnega razvrščanja znotraj okna. Razvoj grafičnega vmesnika v okolju Xcode (domorodna iOS aplikacija) je potekal brez omenjene težave v Android studiu.

4.3 Primerjava uporabe SQLite podatkovne baze

Aplikacije so podatke shranjevale s pomočjo podatkovne baze SQLite. V tem poglavju bomo predstavili razlike implementacije in uporabe te baze pri posameznih razvojnih okoljih. Ob izbrisu aplikacije z mobilne naprave se baza izbriše in podatki so izgubljeni.

Podatkova baza v aplikaciji Cost Note je vsebovala dve ključni tabeli. Prva je bila namenjena shranjevanju podatkov o uporabniku, druga pa shranjevanju podatkov o stroških. Zaradi več vrst stroškov nekatera polja v tabeli stroškov dovolijo prazno vrednost (null).

4.3.1 Android studio

Podpora podatkovni bazi SQLite je privzeto podprta v operacijskem sistemu Android. Potrebno je bilo ustvariti ustrezen razred, ki deduje razred SQLiteOpenHelper. Razvijalec lahko potrebne metode za upravljanje z bazo po želji implementira. Zaradi dedovanja razreda SQLiteOpenHelper je na njegovo zahtevo treba implementirati konstruktor ustvarjenega razreda in metode onCreate ter onUpgrade.

Ustvarili smo razred DB_Controller, ki je javanska datoteka in deduje iz razreda SQLiteOpenHelper (Slika 4.22). V konstruktorju smo poimenovali našo podatkovno bazo CostNote. Ustvarila se bo ob namestitvi aplikacije na mobilno napravo. Akcije nad podatkovno bazo se prožijo s pomočjo metode execSQL, ki za vhodni parameter sprejme SQL poizvedbo v javanskem tipu String. Razvijalec mora za vsako željeno akcijo napisati ustrezno SQL poizvedbo. Pisanje tovrstnih poizvedb je posebnost programskega jezika Java in razvojnega okolja Android studio. Pisanje SQL poizvedb ni potrebno v okoljih Xamarin studio in Xcode.

```java
public class DB_Controller extends SQLiteOpenHelper {

    public DB_Controller(Context context, String name, SQLiteDatabase.CursorFactory factory, int version) {
        super(context, "CostNote.db", factory, version);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL("CREATE TABLE USERS(ID INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE,USERNAME TEXT UNIQUE,PASSWORD TEXT,FULL_NAME TEXT UNIQUE);");
        db.execSQL("CREATE TABLE COST(" +
                "ID_COST INTEGER PRIMARY KEY AUTOINCREMENT UNIQUE," +
                "TYPE TEXT," +
                "DATE_START DATETIME," +
                "DATE_END DATETIME," +
                "COUNTRY_START TEXT," +
                "COUNTRY_END TEXT," +
                "CITY_START TEXT," +
                "CITY_END TEXT," +
                "KILOMETRES_START INT," +
                "KILOMETRES_END INT," +
                "VEHICLE TEXT," +
                "SUM REAL," +
                "IMAGE BLOB," +
                "YEAR INT," +
                "MONTH INT," +
                "STATUS TEXT," +
                "TRANSPORT_TYPE TEXT," +
                "STIMULATION REAL," +
                "USERNAME TEXT);");
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("DROP TABLE IF EXISTS USERS;");
        db.execSQL("DROP TABLE IF EXISTS COST;");
    }
```

Slika 4.22: Implementacija DB_Controller razreda v Javi.

Metoda onCreate se izvede samo ob prvi namestitvi aplikacije. V njej smo ustvarili željene ciljne tabele s pomočjo dveh SQL poizvedb. Metode onUpgrade nismo uporabljali, vendar je bila zaradi dedovanja razreda SQLiteOpenHelper implementacija potrebna. V razredu DB_Controller smo implementirali poljubne metode, ki so nam služile za uporabo podatkovne baze. Na sliki (Slika 4.23) so vidne metode, uporabljene za upravljanje s podatki uporabnika.

```java
public void insert_user(String userName,String password,String fullName){
    ContentValues cv = new ContentValues();
    cv.put("USERNAME",userName);
    cv.put("PASSWORD", password);
    cv.put("FULL_NAME", fullName);
    this.getWritableDatabase().insertOrThrow("USERS", "", cv);
}

public void delete_user(String userName){
    this.getWritableDatabase().delete("USERS", "USERNAME='" + userName + "'", null);
}

public void delete_cost(int id_cost){
    this.getWritableDatabase().delete("COST", "ID_COST='" + id_cost + "'", null);
}

public void update_user(String userName,String password,String fullName,String oldUserName){
    this.getWritableDatabase().execSQL("UPDATE USERS SET USERNAME='" + userName + "', " + "PASSWORD='" + password + "', " +
            "FULL_NAME='" + fullName + "' WHERE USERNAME='" + oldUserName + "';");
}
```

Slika 4.23: Metode v razredu DB_Controller za upravljanje s podatki uporabnika.

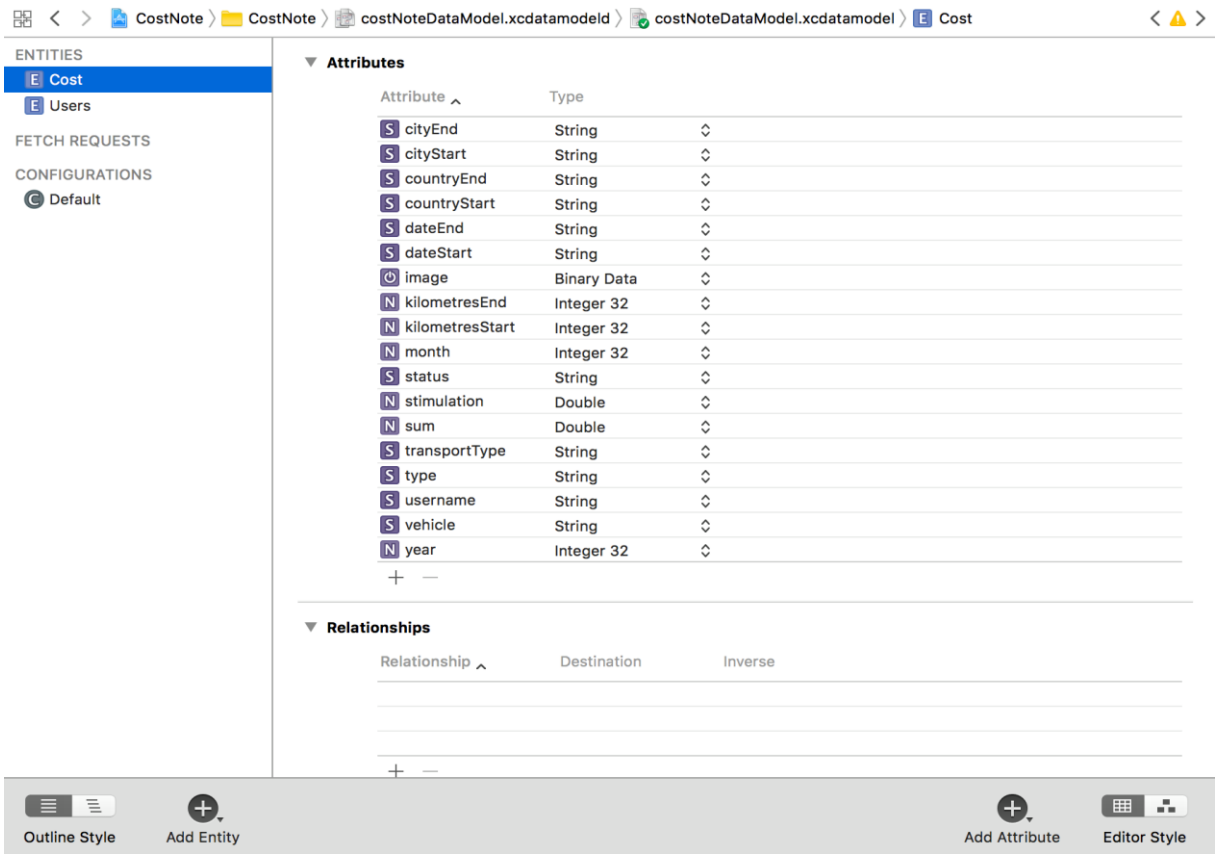
V javanski datoteki, ki služi kot krmilnik enega okna aplikacije, je potrebno inicializirati razred DB_Controller, ki upravlja s podatkovno bazo. Sledi uporaba ustrezne metode. Primer je viden na spodnji sliki (Slika 4.24).

```
//Inicializacija razreda DB_Controller
DB_Controller controllerDB = new DB_Controller(this,"",null,1);

//Uporaba metode za dodajanje novega uporabnika
controllerDB.insert_user(userName.getText().toString(),password.getText().toString(),fullName.getText().toString());
```

Slika 4.24: Inicializacija razreda DB_controller in uporaba metode.

4.3.2 Xcode

Xcode za upravljanje s podatkovno bazo SQLite uporablja ogrodje Core data [2], ki je del razvojnega okolja. V okolju Xcode tabele ustvarimo s pomočjo grafičnega urejevalnika (Slika 4.25). Entiteta predstavlja tabelo. Razvijalec poljubno doda željene atribute na izbrano tabelo. Urejevalnik se odpre s klikom na datoteko vrste xdatamodel v strukturi projekta.



Slika 4.25: Urejevalnik tabel za podatkovno bazo v ogrodju Core data.

Model tabel se v programskem jeziku Swift preslika v objekte z ustreznimi atributi. Razvijalcu metode za ustvarjanje, posodabljanje ali brisanje objekta ni potrebno implementirati. Ob uporabi ene od teh metod mora paziti na ustrezne vrednosti atributov tega objekta (pravi tipi v programskem jeziku). Primera povezave do baze in dodajanja uporabnika sta vidna na spodnji sliki (Slika 4.26).

```
//Povezava do baze
let appDelegate = UIApplication.shared.delegate as! AppDelegate
let context = appDelegate.persistentContainer.viewContext

let request = NSFetchRequest<NSFetchRequestResult>(entityName: "Users")
request.returnsObjectsAsFaults = false

//Nov objekt vrste uporabnik, ki bo dodan v tabelo uporabnikov
let user = NSEntityDescription.insertNewObject(forEntityName: "Users", into: context)

let userFromUI = tvUserName.text
let passFromUI = tvPassword.text
let nameFromUI = tvFullName.text

//Nastavimo podatke za shranjevanje
user.setValue(nameFromUI, forKey: "fullName")
user.setValue(userFromUI, forKey: "username")
user.setValue(passFromUI, forKey: "password")

do{
    try context.save()
    tvUserName.text = ""
    tvPassword.text = ""
    tvFullName.text = ""
    Toast(text: "Uporabnik uspešno shranjen.").show()

}catch{
    fatalError("Napaka pri shranjevanju uporabnika:\(error)")
}
```

Slika 4.26: Dodajanje uporabnika z ogrodjem Core data.

Nad objektom entitete uporabnik smo atributom nastavili določene vrednosti. Razvojno okolje zazna, če se tipi vrednosti ujemajo in na takšen način pomaga razvijalcu. Če vrednosti atributov imena, uporabniškega imena in gesla niso besedilo, okolje na napako opozori. Z metodo Save ogrodje shrani objekt v tabelo. Vsak objekt ima svojo unikatno oznako, ki je kot atribut v tabeli (id) ni bilo potrebno opredeliti. Ta atribut smo v Android studiu opredelili in uporabljali za iskanje ali posodabljanje določenega zapisa.

4.3.3 Xamarin studio

Implementacija podatkovne baze SQLite v okolju Xamarin studio je zahtevala dodajanje zunanjih knjižnic preko čarovnika Nuget, ki implementirajo podatkovno bazo SQLite. Knjižnice smo dodali v vse tri projekte (projekt z deljeno kodo, Xamarin.iOS in Xamarin.Android). Povezava do podatkovne baze mobilne naprave je specifična lastnost posamezne platforme (Android in iOS). V projektu deljene kode smo opredelili vmesnik (Slika 4.27), ki pridobi povezavo do baze.

```
using System;
using SQLite;

namespace CostNote
{
    public interface ISQLite
    {
        SQLiteConnection GetConnection();
    }
}
```

Slika 4.27: Vmesnik v deljenem projektu.

V projektu Xamarin.Android in Xamarin.iOS smo ustvarili razreda Sqlite_Android (Slika 4.28) in Sqlite_iOS (Slika 4.29), ki dedujeta vmesnik deljenega projekta.

```
using System;
using System.IO;
using CostNote.Droid;
using SQLite;
using Xamarin.Forms;

[assembly: Dependency(typeof(Sqlite_Android))]
namespace CostNote.Droid
{
    public class Sqlite_Android : ISQLite
    {
        public Sqlite_Android()
        {
        }

        public SQLiteConnection GetConnection()
        {
            var sqliteFilename = "CostNote.db3";
            string documentsPath = System.Environment.GetFolderPath(System.Environment.SpecialFolder.Personal); // Documents folder
            var path = Path.Combine(documentsPath, sqliteFilename);
            // Create the connection
            var conn = new SQLite.SQLiteConnection(path);
            // Return the database connection
            return conn;
        }
    }
}
```

Slika 4.28: Razred Sqlite_Android v Xamarin.Android projektu.

```
using System;
using System.IO;
using CostNote.iOS;
using SQLite;
using Xamarin.Forms;

[assembly: Dependency(typeof(Sqlite_iOS))]
namespace CostNote.iOS
{

    public class Sqlite_iOS : ISQLite
    {
        public Sqlite_iOS()
        {
        }

        public SQLiteConnection GetConnection()
        {
            var sqliteFilename = "CostNote.db3";
            string documentsPath = Environment.GetFolderPath(Environment.SpecialFolder.Personal); // Documents folder
            string libraryPath = Path.Combine(documentsPath, "..", "Library"); // Library folder
            var path = Path.Combine(libraryPath, sqliteFilename);
            // Create the connection
            var conn = new SQLite.SQLiteConnection(path);
            // Return the database connection
            return conn;
        }
    }
}
```

Slika 4.29: Razred Sqlite_iOS v projektu Xamarin.iOS.

V razredu smo nastavili ustrezno pot do podatkovne baze za obe ciljni platformi. Pot je pomembna za ustvarjanje in dostopanje do podatkovne baze CostNote znotraj mobilne naprave.

Kot pri razvojnem okolju Android studio je v deljenem projektu potrebno ustvariti razred, ki bo imel metodo za dostop in obdelavo baze. Razred smo poimenovali DatabaseAccess. V njem smo vzpostavili povezavo do podatkovne baze. Povezava je odvisna od specifične platforme. Zaradi vmesnika, ki ga dedujeta obe aplikaciji (Xamarin.Android in Xamarin.iOS), bo povezava pravilno ustvarjena. V deljenem projektu smo ustvarili dva dodatna razreda, ki predstavljata tabeli uporabnika in stroška. Razreda imata opredeljene vse potrebne atribute. V okolju Xcode je za razvijalca to storilo ogrodje Core data. V razredu DatabaseAccess smo ustvarili dve zbirki, ki predstavljata tabeli (Slika 4.30).

```
using System;
using SQLite;
using System.Collections.Generic;
using Xamarin.Forms;
using System.Collections.ObjectModel;
using System.Linq;
namespace CostNote
{
    public class DatabaseAccess
    {

        private SQLiteConnection database;
        public ObservableCollection<Cost> costs { get; set; }
        public ObservableCollection<User> users { get; set; }

        public DatabaseAccess()
        {
            database = DependencyService.Get<ISQLite>().GetConnection();
            database.CreateTable<Cost>();
            database.CreateTable<User>();

            if (database != null)
            {
                costs = new ObservableCollection<Cost>(database.Table<Cost>());
                users = new ObservableCollection<User>(database.Table<User>());
            }

        }
```

Slika 4.30: Konstruktor razreda DatabaseAccess.

V tem razredu smo implementirali vse potrebne metode. Na spodnji sliki (Slika 4.31) so vidne metode, ki smo jih uporabili za upravljanje z uporabnikovimi podatki.

```
public void saveUser(string fullName, string username, string password)
{
    database.Insert(new User(fullName, username, password));
}

public void updateUser(User userInstance) {
    database.Update(userInstance);
}

public User getUser(int userId) {
    User getFromDb = users.Where(x => x.UserId == userId).FirstOrDefault();
    return getFromDb;
}
```

Slika 4.31: Metode za upravljanje z uporabnikovimi podatki v razredu DatabaseAccess.

Instanca podatkovne baze najde ustrezno tabelo glede na vrsto objekta (uporabnik ali strošek). To je vidno v metodi saveUser. Vsak objekt ima svoj ključ (id), po katerem ga okolje najde. Iskanje posameznega zapisa po bazi je preprosto zaradi podpore Linq v programskem jeziku C#. Primer iskanja je viden v metodi getUser. Linq je način pisanja poizvedb v programskem jeziku za iskanje izbranega zapisa znotraj vseh podprtih podatkovnih struktur programskega jezika C#.

V krmilniku pogleda vzpostavimo povezavo na bazo in dostopamo do ustrezne metode razreda DatabaseAccess (Slika 4.32).

```
DatabaseAccess database = new DatabaseAccess();
database.saveUser(tvFullName.Text, tvUser.Text, tvPass.Text);
```

Slika 4.32: Povezava do baze in uporaba metode DatabaseAcces.

4.3.4 Ugotovitve

V okoljih je bilo podprto vse, kar smo potrebovali za izbrane ciljne funkcionalnosti aplikacije. V Android studiu je bilo nekoliko moteče pisanje SQL poizvedb za implementacijo željene metode. Ostali dve okolji sta akcije izvajali s pomočjo objektov in tako prihranili čas. V primeru tipkarske napake v SQL poizvedbi določene metode se je delovanje aplikacije ustavilo. Te napake v okolju Xcode in Xamarin studio niso možne. Največ dela je pri povezavi do podatkovne baze zahteval razvoj v Xamarin studiu. Dodatno smo namreč morali razviti vmesnik in razrede tabel. Okolje Xcode je bilo pri hitrosti razvoja najučinkovitejše in je zahtevalo najmanj dela. Velika prednost okolja Xcode je grafični urejevalnik entitet. Razvoj

metod in njihova uporaba so bila v okolju Xamarin studio najbolj prijetna zaradi podpore Linq v programskem jeziku C#. Najslabše se je odrezalo okolje Android studio. Enakovredno sta se odrezali okolji Xcode in Xamarin studio.

4.4 Primerjava hitrosti razvoja aplikacije med razvojnimi okolji

Programska jezika C# in Java nam ob začetku projekta nista bila tuja, programski jezik Swift pa je bil povsem nova izkušnja. To dejstvo je razvoj aplikacije v razvojnem okolju Xcode upočasnilo. Največ težav smo pri razvoju videza aplikacije imeli v okolju Android studio. Uporabniška izkušnja je bila tu v primerjavi z ostalima dvema razvojnima okoljema najslabša. Postavitev nekaterih komponent na okno je velikokrat vplivalo na postavitev že postavljene komponente v obstoječem oknu. Omenjeni problemi so upočasnili razvoj. Videz aplikacije smo najhitreje razvili v okolju Xcode. Logika aplikacije je bila najhitreje razvita v Xamarin studiu. Razvoj grafičnega vmesnika v Xamarin studiu je zaradi nepoznavanja XAML sprva potekal počasi, kasneje pa se je znatno pohitril.

Aplikacija je bila najhitreje razvita v okolju Xamarin studio. Sledil je razvoj aplikacije v okolju Xcode. Najpočasneje smo aplikacijo razvili v okolju Android studio.

4.5 Uporabljene zunanje knjižnice za razvoj aplikacije

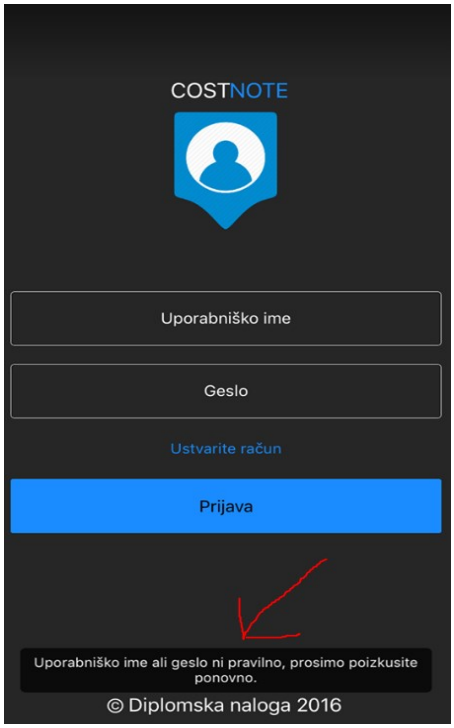
V poglavju bomo našteli vse uporabljene zunanje knjižnice, potrebne za razvoj predpisanih funkcionalnosti aplikacije v posameznih okoljih.

4.5.1 Android studio

V okolju Android studio zunanje knjižnice nismo potrebovali. Vse potrebno je že vključeno v Android platformi.

4.5.2 Xcode

V projekt smo dodali Toast knjižnico [21], ki je namenjena prikazu sporočil za uporabnika. Sporočila smo potrebovali za obvestila o nepravilnem vnosu podatkov. Potrebna dokumentacija in uporaba je prikazana na github spletni strani te knjižnice, ki smo jo prenesli z njihove spletne strani in jo ročno vključili v naš projekt. Knjižnica je odprtokodna. Primer obvestila je viden na spodnji sliki (Slika 4.33).



Slika 4.33: Primer obvestila knjižnice Toast v okolju Xcode.

4.5.3 Xamarin studio

Za uporabo podatkovne baze SQLite je bilo potrebno v projekt vključiti več knjižnic. Vključene knjižnice so SQLitePCLRaw.budnle_green, SQLitePCLRaw.core in sqlite-net-pcl. Za kamero mobilne naprave smo uporabili zunanjo odprtokodno knjižnico Xam.Plugin.Media. Ob dodajanju knjižnice Xam.Plugin.Media smo dobili tudi navodila za nastavitev pravic, potrebnih za uporabo kamere na mobilni napravi. Vse knjižnice smo dodali s pomočjo Nuget managerja. Knjižnice smo dodali v glavni projekt z deljeno kodo in v oba podprojekta (Xamarin.Andorid in Xamarin.iOS).

4.5.4 Ugotovitve

Uporaba zunanjih knjižnic v projektu ni slabost določenega okolja, je pa velika prednost dobra podpora. Razvijalci velikokrat naletijo na določeno tehnično težavo ali problem, ki je že bil rešen. Ob uporabi zunanjih knjižnic je potrebno paziti na licenco. Nekatere knjižnice niso brezplačne in jih je za uporabo v komercialne namen potrebno kupiti. Vse uporabljene knjižnice ciljne aplikacije so brezplačne in odprtokodne. Odprtokodnim knjižnicam lahko razvijalec po želji doda del kode ali po potrebi spremeni obstoječo. Kupljene knjižnice v veliki večini niso odprtokodne.

4.6 Povezovanje grafičnega vmesnika (pogleda) z logiko (krmilnik)

V poglavju bomo prikazali način povezovanja komponent grafičnega vmesnika z logiko (Poslušalci). Vsaka komponenta pogleda, ki ima neko obnašanje na osnovi uporabnikovih odločitev, mora biti znana svojemu krmilniku. Na krmilniku so opredeljene vse potrebne metode in Poslušalci.

4.6.1 Android studio

Datoteko, vrste Java, namenjeno krmilniku določenega pogleda, je bilo potrebno nastaviti v datoteki Manifest. S to nastavitvijo smo aplikaciji dodali novo okno, do katerega lahko dostopa. Primer nastavitve krmilnika je viden spodnji sliki (Slika 4.34). Ime ChoseCost v primeru predstavlja ime razreda namenjenega za krmilnik.

```
<activity android:name=".ChoseCost"
    android:theme="@style/AppTheme.CustomTheme">
</activity>
```

Slika 4.34: Nastavitev novega krmilnika v datoteki Manifest.

V konstruktorju krmilnika izberemo pogled, ki ga želimo prikazati na oknu aplikacije. Pogled smo razvili neodvisno od krmilnika in se po potrebi lahko uporabi večkrat. Primer izbire pogleda je viden na spodnji sliki (Slika 4.35).

```
setContentView(R.layout.activity_chose_cost);
```

Slika 4.35: Nastavite izbranega pogleda za prikaz v oknu krmilnika.

Ob nastavitvi pogleda lahko krmilnik dostopa do vseh komponent na izbranem pogledu. Potrebno je inicializirati željeno komponento in ji nastaviti Poslušalca. Primer Poslušalca za komponento Gumb je viden na spodnji sliki (Slika 4.36). Postopek se ponavlja in je enak za vse komponente. Komponento najdemo s pomočjo unikatne lastnosti id.

```
//Poiščemo komponento na pogledu
btnClosePopUp = (Button) findViewById(R.id.btnChoseCancel);

//Nastavimo logiko v poslušalcu na klik komponente
btnClosePopUp.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent i = new Intent(ChoseCost.this, MainActivity.class);
        i.putExtra("USERNAME",username);
        startActivity(i);
    }
});
```

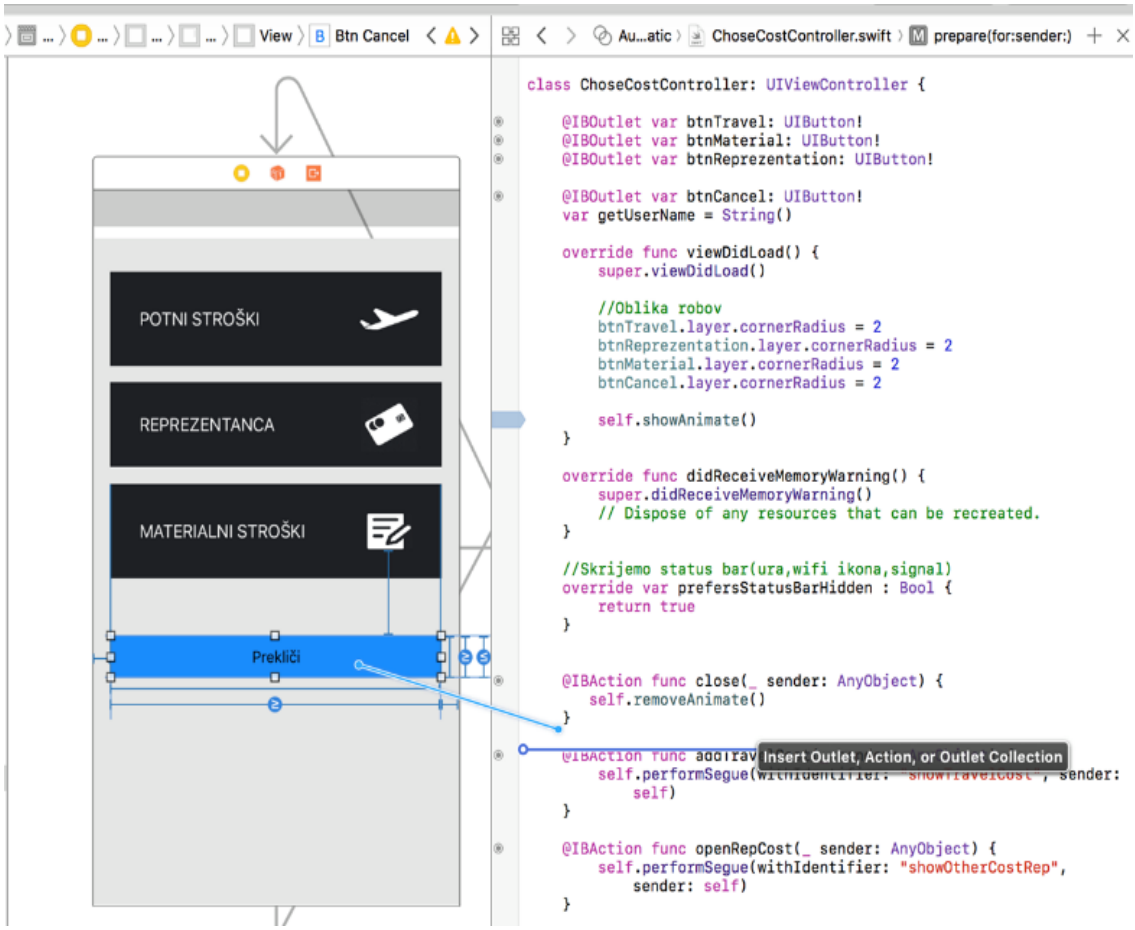
Slika 4.36: Primer Poslušalca za komponento Gumb.

4.6.2 Xcode

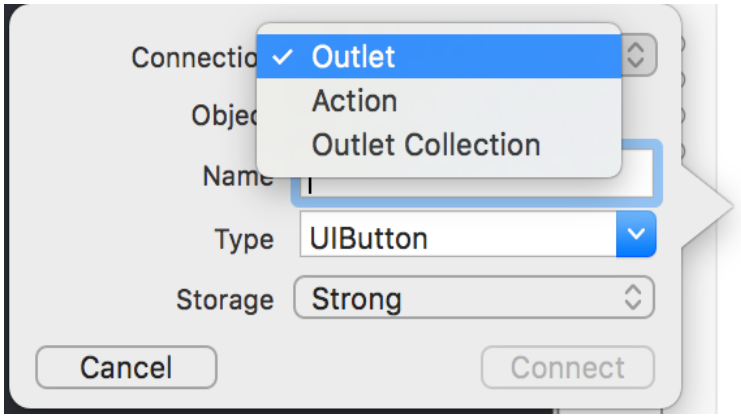
V grafičnem urejevalniku videza aplikacije je bilo oknu potrebno nastaviti ustrezni krmilnik (Swift datoteka). To smo storili v oknu za lastnosti komponent (Slika 4.37).

Slika 4.37: Nastavljanje ustreznega krmilnika.

Z izbranim krmilnikom lahko razvijalec začne z uporabo urejevalnika Pomočnik, preko katerega z metodo povleci in spusti (angleško drag & drop) inicializira komponento in opredeli ustreznega Poslušalca. Ob izbrani komponenti na urejevalniku grafičnega vmesnika s pomočjo tipke cntr (control) in potega miške na datoteko krmilnik (Slika 4.38) razvojno okolje prikaže dialog za izbiro bodisi inicializacije komponente bodisi opredelitev Poslušalca (Slika 4.39).



Slika 4.38: Prikaz metode povleci in spusti (angleško drag & drop) povezovanja komponente s krmilnikom.



Slika 4.39: Dialog za izbiro incializacije komponente ali opredelitev komponente Poslušalca.

Kot primer je na sliki (Slika 4.40) prikazana komponenta Gumb in njen Poslušalec. Postopek se ponavlja čez celoten razvoj aplikacije.

```
//Inicializiramo komponento v razredu krmilnika
@IBOutlet var btnCancel: UIButton!

//Metoda, ki se izvede ob kliku na komponento
@IBAction func close(_ sender: AnyObject) {
    self.removeAnimate()
}
```

Slika 4.40: Inicializacija komponente Gumb in prikaz Poslušalca.

4.6.3 Xamarin studio

Ko v projekt dodamo novo okno, se ustvarita hkrati pogled (XAML datoteka) in krmilnik (C# datoteka). Krmilnik se bo zavedal vseh komponent na pogledu. Razvijalcu pogleda in krmilnika ni potrebno povezati kot pri ostalih dveh okoljih. Komponentam je potrebno nastaviti lastnost x:Name. Preko te lastnosti razvijalec dostopa do komponente na krmilniku, ki je ni potrebno dodatno inicializirati. Razvijalec mora nastaviti Poslušalca in opredeliti ustrezno metodo (Slika 4.41).

```
btnRegister.Clicked += BtnRegister_Clicked;

//Metoda, ki se izvede ob kliku
void BtnCanel_Clicked(object sender, EventArgs e)
{
    Navigation.PushModalAsync(new NavigationPage(new Main(getUserId)));
}
```

Slika 4.41: Prikaz nastavitve Poslušalca in njegova metoda.

Na zgornjem primeru komponenti Gumb določimo ime btnCancel. Ob izbiri komponente se izvede opredeljena metoda. Ta postopek se ponavlja čez celo aplikacijo.

4.6.4 Ugotovitve

Povezovanje krmilnika in pogleda je zahtevalo največ dela v Android studiu. Znotraj okolja Xcode razvijalec s pomočjo urejevalnika lastnosti komponent željenemu pogledu določi ustrezen krmilnik. Najmanj dela za povezovanje pogleda in krmilnika ima razvijalec v okolju Xamarin studio. Razvojno okolje v celoti opravi povezovanje . Implementacija Poslušalcev je med okolji zelo podobna. V okoljih Xamarin studio in Android studio je imel razvijalec enako količino dela. Najhitreje in najpreprosteje je implementirati Poslušalca v okolju Xcode zaradi urejevalnika asistenta in metode povleci in spusti (angleško drag & drop).

4.7 Primerjava implementacij glavnih funkcionalnosti in posebnosti v posameznem okolju

Poglavje je namenjeno predstavitvi implementacije glavnih funkcionalnosti ciljne aplikacije in prikazu nekaterih posebnosti znotraj posameznega razvojnega okolja. Osnovne funkcionalnosti aplikacije so prehodi med različnimi okni (navigacija), komponenta Pogled seznam (seznam stroškov uporabnika), uporaba kamere in prikaz sporočil uporabniku. V poglavju bomo te funkcionalnosti predstavili za vsako razvojno okolje posebej. Zaradi željenega podobnega videza aplikacije smo v Xamarin studiu uporabili »custom renderje«. Pojavna okna v okolju Xcode so bila prav tako razvita zaradi željenega končnega videza.

4.7.1 Implementacija prehodov med okni aplikacije

Prehod med okni aplikacije je osnovna lastnost vsake aplikacije. Na krmilniku, ki je eno okno aplikacije, razvijalec implementira metodo, ki odpre novo okno. Ob odpiranju novega okna pogosto potrebujemo določen podatek iz trenutnega prenesti v novega. Sledil bo prikaz prehoda med okni z nekaterimi dodatnimi podatki za vsako razvojno okolje.

4.7.1.1 Android studio

Intent razred je v programskem jeziku Java eno okno aplikacije. Razvijalec mora Intentu določiti ustrezni razred ciljnega krmilnika, ki ga prikaže v novem oknu. S pomočjo metode putExtra lahko razvijalec poljuben podatek trenutnega okna pošlje v novo okno. Z metodo startAcitivity aplikacija odpre novo okno in zapre trenutnega. Primer odprtja novega okna aplikacije je viden na sliki (Slika 4.42).

```
Intent i = new Intent(getBaseContext(),MainActivity.class);
i.putExtra("USERNAME",userName.getText().toString());
startActivity(i);
```

Slika 4.42: Primer novega okna z dodatnim podatkom.

Na prikazan način (Slika 4.43) lahko na krmilniku novega okna v naslednjem koraku dostopamo do poslanega dodatnega podatka.

```
username = getIntent().getStringExtra("USERNAME");
```

Slika 4.43: Pridobivanje poslanega dodatnega podatka.

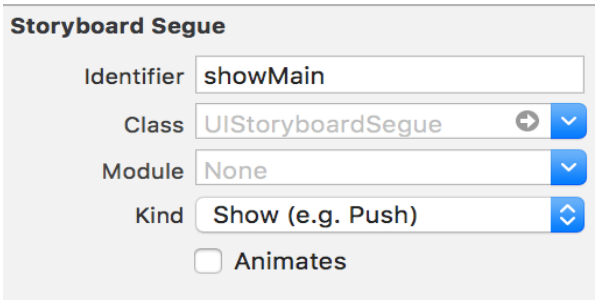
Dodatni podatki so shranjeni v podatkovni strukturi Slovar. Preko ključa dobimo ustrezen podatek. Podatki so tipizirani v programskem jeziku Java. Ob pridobivanju podatka je potrebno paziti na pravo klicano metodo (ustrezen tip). V zgornjem primeru smo v Slovarju uporabniško ime poiskali preko ključa »USERNAME«, ki je besedilo (string v Javi).

4.7.1.2 Xcode

Videz in potek aplikacije je bil razvit s pomočjo Storyboarda okolja Xcode. Storyboard aplikacije je viden na spodnji sliki (Slika 4.44).

Slika 4.44: Prikaz Storyboarda končne aplikacije.

Na Storybordu so vidna vsa okna aplikacije. Smernice oken predstavljajo Segue (povezava), ki povejo, katero okno lahko odpremo iz trenutnega izbranega (navigacija). Za potrebno preusmeritev na novo okno mora razvijalec dodati ustrezen Segue v Storyboard. V urejevalniku lastnosti se določi unikatno ime, ki zaznamuje to povezavo (Slika 4.45). Preko imena odpremo pravilno okno aplikacije. Eno okno ima lahko poljubno število Seguejev. Pojavna okna lahko uporabijo lastnost Storyboard ID (brez povezave Seque), preko katere lahko dostopamo do pojavnega okna v trenutno aktivnem oknu aplikacije.



Slika 4.45: Nastavitev Segueja v Storyboardu.

Vsak krmilnik ima metodo prepere, ki jo deduje iz razreda UIViewController. Razvijalec lahko v tej metodi pošlje dodatne podatke na novo okno (krmilnik). Z metodo performSegue začnemo začetni korak odprtja novega okna aplikacije (Slika 4.46).

```
performSegue(withIdentifier: "showMain", sender: self)
```

Slika 4.46: Klic določenega Sequeja.

Ob klicu Sequeja se v krmilniku začne izvajati koda v metodi prepere (Slika 4.47).

```
override func prepare(for segue: UIStoryboardSegue, sender:
    Any?) {
    if(segue.identifier == "showMain"){
        let DestViewController : MainController = segue.
            destination as! MainController
        DestViewController.getUserName = tvUsername.text!
    }

}
```

Slika 4.47: Primer metode prepere.

V zgornjem primeru DestViewControler postane ciljni krmilnik novega okna. Ko krmilnik incializiramo, dobimo dostop do vseh njegovih spremenljivk. Spremenljivki getUserName smo nastavili uporabniško ime trenutnega okna in tako prenesli podatek.

4.7.1.3 Xamarin studio

Vsak krmilnik ima dostop do globalne spremenljivke Navigation. Gre za podatkovno strukturo seznama, ki hrani objekte razreda NavigationPage. Ob dodajanju novega elementa v vrsto se odpre novo dodano okno. V seznam je potrebno dodati razred krmilnika željnega pogleda. Novi krmilnik dodamo s pomočjo metode PushModalAsync (Slika 4.48).

```
Navigation.PushModalAsync(new NavigationPage(new Main(getUserId)));
```

Slika 4.48: Dodajanje novega krmilnik v seznam Navigation za preusmeritev na novo okno.

Za pošiljanje dodatnih podatkov smo v razred krmilnika dodali ustrezne argumente. Razred smo v naslednjem koraku inicializirali z željenimi podatki. V zgornjem primeru smo ustvarili novi razred Main z argumentom getUserId, ki je unikatna številka uporabnika in preko katere prikazujemo podatke na željenem krmilniku.

4.7.1.4 Ugotovitve

Prehodi med okni aplikacije se v implementaciji znotraj okolij povsem razlikujejo. V razvojnih okoljih Android studio in Xamarin studio lahko iz posameznega okna dostopamo do poljubnega okna aplikacije. V razvojnem okolju Xcode lahko okno dostopa samo do okna, do katerega ima povezavo (Segue v Storyboardu). Xamarin studio se je najboljše izkazal pri enostavnosti pošiljanja podatkov in implementacije.

4.7.2 Implementacija komponente Pogled seznam

Podroben opis delovanja komponente Pogled seznam je prikazan v poglavju 4.1.5. To podpoglavje ja namenjeno prikazu implementacije te komponente v posameznih razvojnih okoljih. Razlog je videz celice seznama za prikaz stroška. Privzeti videz ni zadostil potrebam željenega prikaza posamezne celice.

4.7.2.1 Android studio

Za prikaz željenega videza posamezne celice smo v Android okolje dodali tri nove Java razrede in eno datoteko XML, v kateri smo videz celice razvili s pomočjo grafičnega urejevalnika videza aplikacije. Razred CostAdapter deduje razred ArrayAdapter, ki je zadolžen za vsebino komponente Pogled seznam (podatki celic). Morali smo implementirati metodo getView, ki podatke ustrezno prikaže na komponenti. Slika 4.49 prikazuje implementacijo metode getView.

```
@Override
public View getView(int position,View convertView,ViewGroup parent){
    Cost cost = getItem(position);
    ViewHolder viewHolder;

    //Preveri če je view uporabljen in ponovno uporabi, če ne ustvari novega
    if(convertView == null){
        viewHolder = new ViewHolder(); //shrani reference na polja
        convertView = LayoutInflater.from(getContext()).inflate(R.layout.single_row,parent,false);

        //Mapiranje polj
        viewHolder.type = (TextView) convertView.findViewById(R.id.tvTypeListRow);
        viewHolder.date = (TextView) convertView.findViewById(R.id.tvDateListRow);
        viewHolder.sum = (TextView) convertView.findViewById(R.id.tvSumListRow);
        viewHolder.status = (ImageView) convertView.findViewById(R.id.listRowStatusImg);
        viewHolder.destination = (TextView) convertView.findViewById(R.id.tvDestinationListRow);

        // zapomni si mapiranje če ni novi activity
        convertView.setTag(viewHolder);
    }
    else{
        viewHolder = (ViewHolder) convertView.getTag(); // Pridobi že mapirane reference (Optimizirano)
    }

    //Polnenje mapiranih elementov z ustreznimi podatki
    viewHolder.type.setText(cost.getType());
    viewHolder.date.setText(cost.getDate());
    viewHolder.destination.setText(cost.getDestination());
    viewHolder.sum.setText(cost.getSum() + " €");
    viewHolder.status.setImageResource(cost.getAssociatedDrawable());

    // Vrni vrstico v listu
    return convertView;

}
```

Slika 4.49: Prikaz implementacije metode getView razreda CostAdapter.

V tem razredu smo videz celice nastavili na novoustvarjeno datoteko XML, imenovano single_row. Podatke za prikaz v celici smo ustrezno povezali s komponentami tega pogleda (single_row). Na zgornji sliki je vidna uporaba Razreda Cost (Slika 4.50), ki predstavlja en strošek z vsemi ustreznimi atributi, potrebnimi za prikaz podatkov posamezne celice.

```
public class Cost {
    private double sum;
    private String type,date,destination;

    private long costId,dateCreated;
    private Type typeOfCost;

    public enum Type{APPROVED,PROCESSING}

    public Cost(String type,String date,String destination,double sum,Type typeOfCost,long costId,long dateCreated){
        this.type = type;
        this.date = date;
        this.sum = sum;
        this.typeOfCost = typeOfCost;
        this.dateCreated = dateCreated;
        this.destination = destination;
        this.costId = costId;
    }

    public Cost(String type,String date,String destination,double sum,Type typeOfCost,int costId){
        this.type = type;
        this.date = date;
        this.sum = sum;
        this.typeOfCost = typeOfCost;
        this.dateCreated = 0;
        this.destination = destination;
        this.costId = costId;
    }
}
```

Slika 4.50: Implementacija razreda Cost.

Razred Cost vsebuje dva konstruktorja. Prvi je namenjen prikazu potnega stroška, drugi pa materialnemu in reprezentančnem strošku.

V krmilniku osrednjega okna, ki vsebuje komponento Pogled seznam, uporabimo novonastale razrede (Slika 4.51).

```
//Pridobimo vse trenutne stroške uporabnika iz baze
Cursor allCosts = controllerDB.get_all_cost(username);
costs = new ArrayList<Cost>();

//Sprehodimo se čez vse stroške ter jih dodamo za prikaz
for(allCosts.moveToFirst(); !allCosts.isAfterLast(); allCosts.moveToNext()) {
    Cost.Type type = allCosts.getString(15).equals("APPROVED") == true ? Cost.Type.APPROVED : Cost.Type.PROCESSING;
    if(allCosts.getString(1).equals("POTNI STROŠEK")){
        costs.add(new Cost(allCosts.getString(1),allCosts.getString(3),allCosts.getString(6)+"-"+allCosts.getString(7),
                Double.parseDouble(String.format( "%.2f",allCosts.getDouble(11))),type,allCosts.getInt(0)));
    }
    else {
        costs.add(new Cost(allCosts.getString(1),allCosts.getString(2),"",
                Double.parseDouble(String.format( "%.2f",allCosts.getDouble(11))),type,allCosts.getInt(0)));
    }
}

//Mapiramo za prikaz vseh stoškov
costAdapter = new CostAdapter(getBaseContext(),costs);
costList = (ListView) findViewById(R.id.lvCosts);
costList.setAdapter(costAdapter);
```

Slika 4.51: Implementacija komponente Pogled seznam v krmilniku glavnega okna.

Iz baze smo pridobili vse trenutne stroške uporabnika, ki smo jih nato shranili v podatkovno strukturo Seznam. Ta vsebuje objekte vrste razreda Cost. Ustvarili smo instanco razreda CostAdapter (costAdapter spremenljivka), ki je za argument sprejel seznam Stroškov. Komponenti Pogled seznam smo na osrednjem oknu za vir podatkov nastavili ustvarjeno spremenljivko costAdapter. Celice komponente Pogled seznam so se prikazale v željeni obliki in so vidne na glavnem oknu aplikacije (Slika 4.12).

Nastavili smo Poslušalca na klik posamezne celice in odprli ustrezno okno preko atributa costId razreda Cost (celica).

4.7.2.2 Xcode

Videz celice smo določili v urejevalniku videza aplikacije. Komponenta Pogled seznam vsebuje Prototype Cells celico, ki predstavlja videz vseh celic. V projekt smo dodali razred CustomCell, ki predstavlja en strošek. Vsebuje vse potrebne atribute za prikaz željenih podatkov znotraj celice. Razred deduje lastnosti razreda UITableViewCell, ki predstavlja privzeto eno celico znotraj komponente Pogleda seznam. Implementacija razreda CustomCell je vidna na sliki (Slika 4.52).

```
class customCell: UITableViewCell {

    @IBOutlet var type: UILabel!
    @IBOutlet var date: UILabel!
    @IBOutlet var destination: UILabel!
    @IBOutlet var sum: UILabel!
    @IBOutlet var statusImage: UIImageView!
    var costId = NSManagedObjectID()

    override func awakeFromNib() {
        super.awakeFromNib()
        // Initialization code
    }

    override func setSelected(_ selected: Bool, animated: Bool) {
        super.setSelected(selected, animated: animated)
    }

}
```

Slika 4.52: Implementacija razreda CustomCell.

Krmilnik osrednjega okna je moral dedovati razreda UITableViewDataSoure in UITableViewDelegate. V krmilniku smo ustvarili podatkovno strukturo Seznam, ki hrani podatke o vseh trenutnih stroških prijavljenega uporabnika. Objekti v tem seznamu so razredi vrste NSManagedObject, ki jih uporablja ogrodje Core data (podatkovna baza). Implementacija seznama o stroških je vidna na spodnji sliki (Slika 4.53).

```
//Pridobimo vse trenutne stroške uporabnika
request.returnsObjectsAsFaults = false
request.predicate = NSPredicate(format: "username = %@",getUserName)
let costsFromDb = try context.fetch(request)

if(costsFromDb.count > 0){
    costs = costsFromDb as! Array<NSManagedObject>
}
```

Slika 4.53: Pridobivanje podatkov o vseh stroških uporabnika.

Zaradi dedovanja razreda UITableVIewDataSource smo morali implementirati metodi numberOfRowsInSelection in cellForRowAt. V prvi metodi smo seznamu povedali, koliko celic mora prikazati (število stroškov v seznamu). Druga metoda skrbi za prikazovanje ustreznih podatkov. Slika 4.54 prikazuje implementacijo obeh metod v krmilniku osrednjega okna aplikacije. Aplikacija s pomočjo atributa costId preko Poslušalca na klik posamezne celice uporabnika preusmeri na ustrezno okno. Končni videz seznama je viden na sliki osrednjega okna (Slika 4.12).

```
//MARK: tableView
func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
    return costs.count
}

//MARK: create cell
func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {

    // trenutna celica
    let cell = tableView.dequeueReusableCell(withIdentifier: "cell", for: indexPath) as! customCell

    //Pridobimo podatke iz seznama stroškov
    let getCost = costs[indexPath.row]

    let getSum = getCost.value(forKey: "sum") as? Double
    let getType = getCost.value(forKey: "type") as? String
    let getDate = getCost.value(forKey: "dateStart") as? String
    let getStartCity = getCost.value(forKey: "cityStart") as? String
    let getEndCity = getCost.value(forKey: "cityEnd") as? String
    let getStatus = getCost.value(forKey: "status") as? String

    if(getStatus == "PROCESSING"){
        cell.statusImage.image = UIImage(named: "waiting.png")
    }
    else{
        cell.statusImage.image = UIImage(named: "chekicon.png")
        allSumReturned += getSum!
    }

    //Nastavimo podatke celice
    cell.sum.text = String(getSum!) + " €"
    cell.type.text = getType
    cell.date.text = getDate
    if((getStartCity?.characters.count)! > 0){
        cell.destination.text = getStartCity! + "-" + getEndCity!
    }
    else{
        cell.destination.text = " "
    }

    //id stroška za poslušalca
    cell.costId = getCost.objectID
    return cell
}
```

Slika 4.54: Implementacija potrebnih metod za prikaz podatkov v komponenti Pogled seznam.

Spremenljivka indexPath je številka vseh stroškov v seznamu costs. Za vsak strošek se je ustvarila celica, ki smo ji nastavili ustrezne podatke za prikaz.

4.7.2.3 Xamarin studio

V projekt smo dodali dva nova razreda. Razred CustomCellModel vsebuje vse potrebne atribute za prikaz ustreznih podatkov v željeni celici komponente Pogled seznam. Implementacija razreda CustomCellModel je vidna na spodnji sliki (Slika 4.55).

```
public class CustomCellModel
{
    public string Type { get; set;}
    public string Date { get; set;}
    public string Destination { get; set; }
    public string Sum { get; set; }
    public string Image { get; set; }
    public string CostId { get; set; }
}
```

Slika 4.55: Razred CustomCellModel.

Drugi razred smo poimenovali CustomCostCell in deduje razred ViewCell. Razred je zadolžen za prikaz celice v ustreznem izgledu. V tem okolju je bila posebnost razvoj videza celice v programskem jeziku in ne v urejevalniku XAML. Razred CustomCostCell smo definirali znotraj krmilnika osrednjega okna aplikacije, ki vsebuje komponento Pogled seznam. Implementacija razreda CustomCostCell je vidna na spodnji sliki (Slika 4.56).

```
public class CustomCostCell : ViewCell
{
    public CustomCostCell() {
        Label type = new Label();
        Label date = new Label();
        Label destination = new Label();
        Label sum = new Label();
        Image img = new Image();
        Label costId = new Label();

        img.HorizontalOptions = LayoutOptions.Center;
        img.VerticalOptions = LayoutOptions.Center;
        sum.VerticalTextAlignment = TextAlignment.End;
        type.FontSize = 16;
        date.FontSize = 16;
        destination.FontSize = 16;
        sum.FontSize = 16;
        img.Scale = 1.8;
        costId.IsVisible = false;

        //set bindigs
        type.SetBinding(Label.TextProperty, new Binding("Type"));
        date.SetBinding(Label.TextProperty, new Binding("Date"));
        destination.SetBinding(Label.TextProperty, new Binding("Destination"));
        sum.SetBinding(Label.TextProperty, new Binding("Sum"));
        img.SetBinding(Image.SourceProperty, new Binding("Image"));
        costId.SetBinding(Label.TextProperty, new Binding("CostId"));


          Grid grid = new Grid
          {
              Padding = new Thickness(15),
              RowDefinitions = {
                  new RowDefinition { Height = 17},
                  new RowDefinition { Height = 17},
                  new RowDefinition { Height = 21},
              },
              ColumnDefinitions = {
                  new ColumnDefinition {Width = new GridLength(1,GridUnitType.Star)},
                  new ColumnDefinition {Width = new GridLength(1,GridUnitType.Star)},
                  new ColumnDefinition {Width = new GridLength(1,GridUnitType.Star)},
                  new ColumnDefinition {Width = new GridLength(1,GridUnitType.Star)},
              }
          };

          grid.Children.Add(type, 0, 0);
          Grid.SetColumnSpan(type, 3);
          grid.Children.Add(img, 3, 0);

          grid.Children.Add(date, 0, 1);
          Grid.SetColumnSpan(date, 2);
          grid.Children.Add(destination, 0, 2);
          Grid.SetColumnSpan(destination, 2);
          grid.Children.Add(sum, 3, 2);

          View = grid;
    }
}
```

Slika 4.56: Implementacija razreda CustomCostCell.

Opredelili smo vse ustrezne komponente, ki jih potrebujemo za prikaz v celici. S pomočjo metode SetBindig smo lastnost TextProperty vseh komponent povezali z ustreznimi atributi razreda CustomCellModel.

V krmilniku osrednjega okna aplikacije smo uporabili vse novonastale razrede, potrebne za prikaz celice v ustreznem videzu znotraj komponente Pogled seznam (Slika 4.57).

```
allCosts = new List<CustomCellModel>();
costFromDb = database.getCosts(UserId);

double sumToGet = 0;
double sumGot = 0;

//Sprehodimo se čez vse stroške uporabnika, ter jih dodamo v seznam stroškov namejene prikazu
foreach (var cost in costFromDb) {
    if (cost.Status.Equals("PROCESSING"))
    {
        if (cost.Type.Equals("POTNI STROŠEK"))
        {
            allCosts.Add(new CustomCellModel { Type = cost.Type, Date = cost.DateStart, Destination = cost.CityStart + " - " +
            cost.CityEnd, Sum = (cost.Sum + cost.Stimulanse) + " €", Image = "waiting.png", CostId = cost.CostId.ToString() });
            sumToGet += (cost.Sum + cost.Stimulanse);

        }
        else {
            allCosts.Add(new CustomCellModel { Type = cost.Type, Date = cost.DateStart, Destination = "",
            Sum = (cost.Sum + cost.Stimulanse) + " €", Image = "waiting.png", CostId = cost.CostId.ToString() });
            sumToGet += (cost.Sum + cost.Stimulanse);
        }
    }
    else {
        if (cost.Type.Equals("POTNI STROŠEK"))
        {
            allCosts.Add(new CustomCellModel { Type = cost.Type, Date = cost.DateStart, Destination = cost.CityStart + " - "
            + cost.CityEnd, Sum = (cost.Sum + cost.Stimulanse) + " €", Image = "chekicon.png", CostId = cost.CostId.ToString() });
            sumGot += (cost.Sum + cost.Stimulanse);

        }
        else {
            allCosts.Add(new CustomCellModel { Type = cost.Type, Date = cost.DateStart, Destination = "",
            Sum = (cost.Sum + cost.Stimulanse) + " €", Image = "chekicon.png", CostId = cost.CostId.ToString() });
            sumGot += (cost.Sum + cost.Stimulanse);
        }
    }
}

//Nastavimo podatke za prikaz
costsListView.ItemsSource = allCosts;
costsListView.RowHeight = 90;

//Nastavimo izgled celic
costsListView.ItemTemplate = new DataTemplate(typeof(CustomCostCell));

//Poslušalec na klik celice
costsListView.ItemSelected += CostsListView_ItemSelected;
```

Slika 4.57: Implementacija komponente Pogled seznam.

Vse stroške uporabnika smo shranili v podatkovno strukturo Seznam (allCosts spremenljivka), ki hrani objekte vrste CustomCellModel. Komponenti pogled Seznam smo za vir podatkov nastavili spremenljivko allCosts. Videz posamezne celice smo določili preko lastnosti ItemTemplate (uporaba razreda CustomCostCell). Poslušalca na klik posamezne celice smo implemetirali s pomočjo atributa CostId razreda CustomCostModel. Končni videz komponente Pogled seznam je viden na sliki osrednjega okna aplikacije (Slika 4.12).

4.7.2.4 Ugotovitve

Implementacija celice po meri je bila med okolji dokaj podobna. V vseh treh okoljih je bilo potrebno ustvariti novi programski razred, ki je posamezna celica komponente Pogled seznam. Zaradi programskega razvoja videza je bila implementacija videza celice najpočasnejša v okolju Xamarin studio. Okolji Xcode in Android studio sta se pri razvoju videza posamezne

celice odrezali enakovredno. Najboljša uporabniška izkušnja je bila v razvojnem okolju Xcode (Prototype Cells). Videz celic lahko razvijalec določi neposredno v komponenti Pogled seznam znotraj grafičnega urejevalnika videza aplikacije.

4.7.3 Uporaba kamere

Uporaba kamere mobilne naprave je bila zahteva v funkcionalnih specifikacijah ciljne aplikacije (poglavje 1.1). S kamero smo uporabniku omogočili slikanje »računa« in njegovo hrambo znotraj aplikacije.

4.7.3.1 Android studio

Uporaba kamere je bila podprta znotraj emulatorja mobilne naprave Android. Za implementacijo nismo potrebovali zunanje knjižnice. V datoteki Androidmanifest je bilo treba nastaviti pravice za uporabo vhodnih naprav emulatorja (osebni računalnik). Nastavitev pravic je vidna na spodnji sliki (Slika 4.58).

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-feature android:name="android.hardware.camera" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

Slika 4.58: Pravice za uporabo kamere v datoteki Androidmanifest.

Platforma Android ima s strani uporabnika podprto okno za zajemanje slike, ki sliko vrne v podatkovnem tipu Bitmap. Ta podatkovni tip v programskem jeziku Java predstavlja sliko. Bitmap sliko smo pretvorili v seznam bytov in jo shranili v podatkovno bazo SQLite. Pretvorba je bila nujna, saj podatkovna baza ne pozna tipa Bitmap. Na komponento Pogled slika smo nastavili Poslušalca, ki sproži okno za zajemanje slike (Slika 4.59).

```
takePhoto.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        Intent i = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
        i.putExtra(MediaStore.EXTRA_VIDEO_QUALITY,1);
        startActivityForResult(i,0);
    }
});
```

Slika 4.59: Poslušalec za prikaz Androidovega okna za zajemanje slike.

Podatki zajete slike se shranijo v spremenljivko i. Potrebno je bilo implementirati metodo onActivityResult, ki se sproži ob končanem slikanju. Slika 4.60 prikazuje implementacijo željene metode.

```
@Override
protected void onActivityResult(int requestedCode,int resultCode,Intent data){
    if(requestedCode == 0){
        switch (resultCode){
            case Activity.RESULT_OK:
                Bundle extras = data.getExtras();
                Bitmap imageBitmap = (Bitmap) extras.get("data");
                imageForDb = getBytes(imageBitmap);
                break;
            case Activity.RESULT_CANCELED:
                break;
            default:
                break;
        }
    }
}

// convert from bitmap to byte array
public static byte[] getBytes(Bitmap bitmap) {
    ByteArrayOutputStream stream = new ByteArrayOutputStream();
    bitmap.compress(Bitmap.CompressFormat.PNG, 0, stream);
    return stream.toByteArray();
}
```

Slika 4.60: Implementacija metode onActivityResult okna za zajemanje slike.

Implementirali smo pomožno metodo getBytes, ki Bitmap sliko pretvori v seznam bytov. Spremenljivko imageForDb lahko nato uporabimo za shranjevanje v podatkovno bazo SQLite. Za prikaz slike iz baze smo uporabili metodo decodeByteArray razreda BitmapFactory, podprt v programskem jeziku Java. Metoda pretvori seznam bytov v podatkovni tip Bitmap. Pretvorjeno sliko lahko nato prikažemo v komponenti Pogled slika.

4.7.3.2 Xcode

Uporaba kamere in vhodnih naprav je možna samo na fizičnih mobilnih napravah z iOS operacijskim sistemom. V projektu je bilo uporabljeno okno platforme iOS za zajem slike uporabnika. Zunanja knjižnica ni bila potrebna. V sistemski datoteki Info.plist je bilo potrebno dodati pravico za uporabo kamere (Slika 4.61).

Slika 4.61: Pravice za uporabe kamere znotraj projekta v okolju Xcode.

Krmilnik okna, ki je uporabljal kamero, je moral dedovati razred UIImagePickerControllerDelegate. Implementirati smo morali dedovani metodi imagePickerControllerDidCancel in imagePickerController za odpiranje okna, v katerem uporabnik zajeme sliko. Komponenti Pogled slika smo dodali Poslušalca, ki je sprožil metodo useCamera (Slika 4.62). Znotraj te metode smo prikazali okno za zajem slike.

```
let cameraListner = UITapGestureRecognizer(target: self, action: #selector(TravelCostController.useCamera))
btnFoto.addGestureRecognizer(cameraListner)
```

Slika 4.62: Poslušalec za odpiranje okna zajema slike na platformi iOS.

Komponenta Pogled slika prvotno ne podpira Poslušalca na klik (spremenljivka btnFoto). Implementirali smo ga s pomočjo razreda UITapGestureRecognizer, ki zazna klik na komponento. Implementacija dedovanih razredov je prikazana na spodnji sliki (Slika 4.63).

```
@IBAction func useCamera(){
    let picker = UIImagePickerController()
    picker.delegate = self
    picker.sourceType = .camera
    picker.allowsEditing = true
    picker.mediaTypes = UIImagePickerController.availableMediaTypes(for: picker.sourceType)!
    self.present(picker,animated:true,completion: nil)
}

func imagePickerControllerDidCancel(_ picker: UIImagePickerController) {
    self.dismiss(animated: true, completion: nil)
    print("Kamera zaprta!")
}

func imagePickerController(_ picker: UIImagePickerController, didFinishPickingMediaWithInfo info: [String : Any]) {

    let mediaType = info[UIImagePickerControllerMediaType] as! String

    if mediaType == (kUTTypeImage as String){
        let image = (info[UIImagePickerControllerOriginalImage] as? UIImage)
        // create NSData from UIImage
        imageFromUser = UIImageJPEGRepresentation(image!, 1)!
        isImageTaken = true
    }

    self.dismiss(animated: true, completion: nil)
}
```

Slika 4.63: Implementacija potrebnih metod za uporabo kamere na platformi iOS.

Znotraj metode useCamera smo odprli okno za zajem slike. Ob uspešnem zajemu slike uporabnika se je sprožila metoda imagePickerController. Slika je bila v podatkovnem tipu UIImage. Z metodo UIImageJPEGRepresentation smo sliko pretvorili v seznam bytov. Pretvorjeno sliko lahko shranimo v podatkovno bazo SQLite. Za prikaz slike iz baze pretvorba ni bila potrebna. Komponenta Pogled slika prikaže sliko iz seznama bytov preko lastnosti image.

4.7.3.3 Xamarin studio

Za implementacijo uporabe kamere smo v Xamarin studiu potrebovali zunanjo knjižnico Xam.Plugin.Media [16]. Slike smo shranjevali na trdi disk naprave in ne v podatkovno bazo SQLite. V bazo smo shranili zgolj pot do slike za kasnejši prikaz. Knjižnico smo dodali v vse tri projekte s pomočjo Nuget manegerja. Ob namestitvi knjižnice so bila dodana navodila za

nastavitev. Pravic za dostop do kamere nam v projektu Xamarin.Android ni bilo potrebno nastaviti (knjižnica je to storila za nas). Dodati je bilo potrebno datoteko vrste XML, poimenovano file_paths. V njej smo nastavili mesto shranjevanja zajetih slik (Slika 4.64).

```
<?xml version="1.0" encoding="utf-8"?>
<paths xmlns:android="http://schemas.android.com/apk/res/android">
    <external-path name="my_images" path="Android/data/com.diploma.costnote/files/Pictures" />
    <external-path name="my_movies" path="Android/data/com.diploma.costnote/files/Movies" />
</paths>
```

Slika 4.64: Nastavitev mesta shranjevanja znotraj Android aplikacije.

Mesta shranjevanja v projektu Xamarin.iOS ni bilo potrebno dodati. V datoteki Info.plist smo dodali dve pravici za uporabo kamere in trdega diska mobilne naprave (Slika 4.65).

Privacy - Photo Library Usage Description	String	This app needs access to the camera to take photos.
Privacy - Camera Usage Description	String	This app needs access to the camera to take photos.

Slika 4.65: Pravice za uporabo kamere in trdega diska mobilne naprave.

Znotraj krmilnika smo implementirali zgolj Poslušalca na komponento Pogled slika (spremenljivka btnFoto), ki je predstavljala gumb za zajem slike (Slika 4.66). Podobno kot pri okolju Xcode je implementacija Poslušalca na komponento Pogled slika nekoliko drugačna, vendar trivialna.

```
btnFoto.GestureRecognizers.Add(new TapGestureRecognizer
{
    Command = new Command(async () =>
    {
        await CrossMedia.Current.Initialize();

        if (!CrossMedia.Current.IsCameraAvailable || !CrossMedia.Current.IsTakePhotoSupported)
        {
            await DisplayAlert("No Camera", ":( No camera available.", "OK");
            return;
        }

        var file = await CrossMedia.Current.TakePhotoAsync(new Plugin.Media.Abstractions.StoreCameraMediaOptions
        {
            Directory = "Sample",
            Name = database.getIdForPicture(userId).ToString() + "-" + database.getUser(userId).UserName + "-racun.png"
        });

        if (file == null)
            return;

        //await DisplayAlert("File Location", file.Path, "OK");
        imageFromUser = file.Path;
    }),
    NumberOfTapsRequired = 1
});
```

Slika 4.66: Uporaba zunanje knjižnice za zajem slike.

Knjižnica odpre privzeto okno za zajem slike specifične platforme (Android ali iOS). Okno za zajem slike je enako kot pri domorodnih aplikacijah. V bazo SQLite smo shranili pot do zajete slike. Za prikaz slike na komponenti Pogled slike smo uporabili lastnost Source. Kot vrednost prejme pot do slike in jo prikaže.

4.7.3.4 Ugotovitve

Razvijalcem brez mobilne naprave z operacijskim sistemom iOS ne bi bilo mogoče testirati funkcionalnost kamere. Emualtor namreč tega ne omogoča. Gre za moteč dejavnik pri razvoju testnih aplikacij v primerjavi z Android platformo. Razvojno okolje Xamarin studio je v primerjavi z Xcode in Android studiem potrebovalo uporabo zunanje knjižnice. Implementacija je bila podobna in pri nobenem koraku izrazito težja. Najhitreje in najpreprosteje smo to storili v Xamarin studiu. Razvoj v Xcode in Android studiu je potekal težavnostno in časovno enakovredno.

4.7.4 Prikaz sporočil uporabniku

Obvestila služijo kot pomoč uporabnikom in so del vseh sodobnih aplikacij. V primeru napačno vnesenih podatkov s strani uporabnika je treba to uporabniku sporočiti preko obvestila znotraj aplikacije. Obvestila smo uporabili tudi za prikaz uspešno dodanega stroška ali uspešne registracije novega uporabnika.

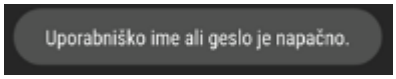
4.7.4.1 Android studio

Platforma Android ima prikazovanje obvestil podprto z uporabo javanskega razreda Toast. Prikazovanje sporočila je zelo preprosto in implementacija je vidna na spodnji sliki (Slika 4.67).

```
Toast.makeText(LoginActivity.this,"Uporabniško ime ali geslo je napačno.",Toast.LENGTH_LONG).show();
```

Slika 4.67: Implementacija prikaza obvestila na platformi Android.

Razvijalec lahko določi časovni prikaz obvestila preko tretjega argumenta metode makeText. Čas obvestila določi, kdaj se to odstrani z vmesnika aplikacije. Obvestilo je lahko poljubne dolžine in vsebine. Videz sporočila na grafičnem vmesniku je viden na spodnji sliki (Slika 4.68).



Slika 4.68: Prikaz obvestila na grafičnem vmesniku Android aplikacije.

4.7.4.2 Xcode

Zaradi zaporednega razvoja in željenega podobnega videza aplikacij smo za prikaz obvestil uporabili zunanjo knjižnico Toast, omenjeno v poglavju 4.5.2. Uporaba knjižnice Toast je vidna na spodnji sliki (Slika 4.69).

```
Toast(text: "Uporabniško ime ali geslo ni pravilno, prosimo poizkusite ponovno.").show()
```

Slika 4.69: Uporaba zunanje knjižnice Toast za prikaz obvestila v okolju Xcode.

Razred Toast ima prav tako možnost nastavljanja časovnega prikaza obvestila in spreminjanje postavitve na grafičnem vmesniku aplikacije. Obvestilo se odstrani po določenem času. Za naše potrebe smo uporabili privzete nastavitve. Videz obvestila je viden na sliki (Slika 4.33).

4.7.4.3 Xamarin studio

Na platformi Xamarin smo uporabili sistemska sporočila za prikaz obvestil uporabniku. Sporočila se med obema platformama po videzu razlikujejo. Prikaz je implementiran v skupnem projektu z deljeno kodo (Slika 4.70).

```
DisplayAlert("Opozorilo", "Neveljaven vpis,prosimo poizkusite ponovno.", "OK");
```

Slika 4.70: Prikaz Implementacije sporočila v deljenem projektu.

Uporabnik mora obvestilo zapreti preko gumba v obvestilu. Videz obvestil v obeh platformah je viden na spodnji sliki (Slika 4.71).

Slika 4.71: Prikaz sistemskih obvestil. Levo (Xamarin.Android) in desno (Xamarin.iOS).

4.7.4.4 Ugotovitve

Uporaba zunanje knjižnice Toast v razvojnem okolju Xcode ni predstavljala dodatne ovire in razvoja ni upočasnila. Uporaba obvestil je bila v vseh okoljih zelo preprosta (ena vrstica kode). Videz obvestil je bil enoten pri obeh domorodnih aplikacijah (uporaba toast sporočil).

Sistemska obvestila so se prav tako preslikala v dokaj enoten videz. Vsa okolja so se pri primerjavi dobro odrezala in ni mogoče izpostaviti izrazitega zmagovalca.

4.7.5 Custom rendererji v platformi Xamarin.Forms

Komponente grafičnega vmesnika na platformi Xamarin.Forms se preslikajo v domorodne komponente specifične platforme (Android in iOS). Zaradi razlik med domorodnimi komponentami Xamarin.Forms komponente podpirajo zgolj osnovne lastnosti komponent. S pomočjo »custom renderejev« lahko razvijalec določi manjkajoče lastnosti in tako po potrebi sam oblikuje videz komponente. Razlog tega podpoglavja je posebnost platforme Xamarin.Forms, ki je ključna za razvoj komercialnih aplikacij. Za prikaz implementacije bomo predstavili »custom renderer« za komponento vnosno polje, ki smo ji dodali belo barvo obrobe in zavite robove.

V deljenem projektu je potrebno ustvariti razred, ki deduje izbrano komponento. Poimenovali smo ga BorderedEntry (Slika 4.72).

```
using System;
using Xamarin.Forms;
namespace CostNote
{
    public class BorderedEntryLogin : Entry
    {
        public BorderedEntryLogin()
        {
        }
    }
}
```

Slika 4.72: Razred BorderedEntry v deljenem projektu.

V podprojektih Xamarin.Andorid in Xamarin.iOS je potrebno ustrezno implementirati komponento BorderedEntry. V datoteki XAML ustreznega pogleda namesto komponente Entry uporabimo komponento BordereEntry (Slika 4.73). Ob prevajanju aplikacije projekt poišče implementacije komponente BorderedEntry v podprojektih in ga ustrezno prikaže na grafičnem vmesniku aplikacije.

```
<local:BorderedEntry Grid.Row="3" Grid.Column="2" Margin="0,0,10,0" Grid.ColumnSpan="2"
x:Name="tvSum" BackgroundColor="#777777" HorizontalTextAlignment="Center" HorizontalOptions="Start" Placeholder="Vnesite znesek"
TextColor="Black" Keyboard="Numeric" PlaceholderColor="Black"></local:BorderedEntry>
```

Slika 4.73: Uporaba BorderEntry razreda v datoteki XAML.

V oba podprojekta smo dodali Razred BorderedEntryRenderer, ki implementira komponento BordererEntry. Implementacija Xamarin.Android je vidna na spodnji sliki (Slika 4.74).

```
using System;
using Xamarin.Forms;
using Xamarin.Forms.Platform.Android;
using Android.Graphics.Drawables;
using CostNote;
using CostNote.Droid;

[assembly: ExportRenderer(typeof(BorderedEntry), typeof(BorderedEntryRenderer))]
namespace CostNote.Droid
{
    public class BorderedEntryRenderer : EntryRenderer
    {
        protected override void OnElementChanged(ElementChangedEventArgs<Entry> e)
        {
            base.OnElementChanged(e);

            if (Control != null)
            {
                Control.SetBackgroundResource(Resource.Drawable.etstyle);
            }
        }
    }
}
```

Slika 4.74: Prikaz razreda BordererEntryRenderer v projektu Xamarin.Android.

Komponenti BordereEntry smo za ozadje nastavili datoteko XML iz domorodne aplikacije Android (beli robovi in oblika robov zaobljena). Slika 4.74 prikazuje implementacijo razreda BorderedEntryRenderer v projektu Xamarin.iOS.

```
using System;
using CostNote;
using CostNote.iOS;
using UIKit;
using Xamarin.Forms;
using Xamarin.Forms.Platform.iOS;
using CoreGraphics;
[assembly: ExportRenderer(typeof(BorderedEntry), typeof(BorderedEntryRenderer))]
namespace CostNote.iOS
{
    public class BorderedEntryRenderer : EntryRenderer
    {
        protected override void OnElementChanged(ElementChangedEventArgs<Entry> e)
        {
            base.OnElementChanged(e);

            if (Control != null)
            {
                //Dostop do vseh lastnsnosti domorodne komponente UITextField.
                Control.Layer.BorderColor = UIColor.White.CGColor;
                Control.Layer.BorderWidth = 0.5f;
                Control.Layer.CornerRadius = 3;
            }
        }
    }
}
```

Slika 4.75: Prikaz razreda BordererEntryRenderer v projektu Xamarin.iOS.

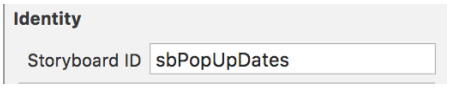
Komponenti smo tako kot pri implementaciji Xamarin.Android nastavili belo barvo robov in zaoblili njihovo obliko.

Oba razreda sta dedovala razred EntryRenderer domorodne aplikacije specifične platforme (Android in iOS). S tem dedovanjem lahko razvijalec dostopa do vseh lastnosti, ki jih komponenta podpira. V zgornjem primeru smo potrebovali le barvo in obliko obrobe vnosnega polja. Končni videz vnosnih polj je viden na sliki (Slika 4.18).

Manjkajoče osnovne lastnosti komponent (barva obrobe vnosnega polja) so moteče. Platforma Xamarin.Forms razvijalcu ponuja razvoj domorodne komponente s pomočjo »custom rendererjev«, vendar predstavlja implementacija nekaj dela. V ciljni aplikaciji smo »custom rendererje« poleg zgoraj omenjenega uporabili še za poravnavo besedila v komponenti Izbirik in poravnavo besedila v Izbirniku datuma.

4.7.6 Pojavna okna v razvojnem okolju Xcode

Pojavna okna smo potrebovali za prikaz komponente Izbirnik in Izbirnik datuma v domorodni aplikaciji iOS. Razvoj domorodne aplikacije iOS je potekal za domorodno Android aplikacijo. Videz smo želeli poenotiti z uporabo pojavnih oken. Okno smo razvili na enak način kot normalno okno (pogled) aplikacije. Dodali smo krmilnik in ustrezni pogled. Pogled okna smo priredili, tako da je izgledalo kot pojavno okno (manjše velikosti). Kot primer implementacije smo prikazali pojavno okno za komponento Izbirnik datuma. Preko urejevalnika lastnosti smo oknu, namenjenem za pojavno okno, določili lastnost Storyboard ID (Slika 4.76).



Slika 4.76: Določanje lastnosti Storyboard ID pogledu pojavnega okna.

Pojavno okno za izbiro datuma je uporabljeno na vseh oknih za vnos stroška. Preko Storyboard ID lastnosti lahko do pojavnega okna dostopamo iz poljubnega okna aplikacije. Pojavno okno potrebuje povezavo (Segue) v starševsko okno (kjer je bil klican). Povezava je potrebna zaradi pošiljanja podatkov. V projekt smo dodali krmilnik z imenom PopUpDatesController. Implementacija tega krmilnika je povsem enaka ostalim. V nadrejenem oknu odpremo pojavno okno (Slika 4.77) s pomočjo Poslušalca na komponenti Gumb. Prehod med normalnimi okni aplikacije uniči prejšnje okno ter sprosti pomnilnik in procesor mobilne naprave. Ob odprtju pojavnega okna se nadrejeno okno preprosto ustavi (miruje), v ospredju deluje pojavno okno, v katerem lahko uporabnik uporablja vse funkcionalnosti. Prehod iz pojavnega okna na nadrejeno okno uniči pojavno okno. Podatki v glavnem oknu se posodobijo in okno preide nazaj v stanje delovanja.

```
@IBAction func selectDate(_ sender: AnyObject) {
    let popOverVC = UIStoryboard(name: "Main", bundle: nil).instantiateViewController
        (withIdentifier: "sbPopUpDates") as! PopUpDatesController
    popOverVC.getUserName = self.getUserName
    popOverVC.whichController = "other"
    popOverVC.getOtherCostTitle = getTitle
    self.addChildViewController(popOverVC)
    self.view.addSubview(popOverVC.view)
    popOverVC.didMove(toParentViewController: self)
}
```

Slika 4.77: Metoda za prikaz pojavnega okna.

Slika 4.17 prikazuje videz pojavnega okna za izbiro datuma. Implementacija pojavnega okna ni predstavljala večjih težav, saj deluje enako kot navadno okno aplikacije. Pri implementaciji je razliko predstavljal le Poslušalec za prikaz pojavnega okna (ne preko Segueja).

Poglavje 5 **Sklepne ugotovitve**

V diplomski nalogi smo primerjali razvojna okolja Android studio, Xcode in Xamarin studio, ki omogočajo razvoj mobilnih aplikacij. Ciljni platformi sta bili Android in iOS. Cilj projekta je bil domorodno aplikacijo izdelati za obe platformi. S pomočjo ogrodja Xamarin.Forms lahko razvijalec v okolju Xamarin studio razvije večplatformsko aplikacijo (Android in iOS hkrati). V diplomski nalogi smo primerjali razvoj aplikacije v domorodnih okoljih in ogrodju Xamarin.Forms. Zanimal nas je časovni razvoj, podpora vsem funkcionalnostim in končni izgled aplikacije. Najprej smo s pomočjo Android studia razvili domorodno aplikacijo za Android. V naslednji fazi smo razvili domorodno aplikacijo iOS v razvojnem okolju Xcode. Večplatformska aplikacija je bila kot zadnja razvita v Xamarin studiu.

Končne aplikacije so bile glede na želene funkcionalnosti uspešno implementirane. Videza aplikacije nismo uspeli v celoti poenotiti. Težave smo imeli pri velikostih in poravnavah komponent grafičnega vmesnika aplikacij. Razvoj aplikacij v posameznih razvojnih okoljih je trajal sorazmerno enako. V razvojnih okoljih smo uporabili drugačne programske jezike in urejevalnik videza. Pri tem so bile skupne lastnosti aplikacij zgolj slike in barve.

V Android studiu smo največ težav imeli z razvrščanjem komponent v urejevalniku grafičnega vmesnika aplikacije. Za razvoj aplikacije nismo potrebovali zunanjih knjižnic, saj je bilo vse potrebno privzeto že podprto na platformi Android. V primerjavi z aplikacijo iOS smo lahko vse funkcionalnosti testirali na emulatoju (kamera).

Najboljša uporabniška izkušnja razvoja grafičnega vmesnika je bila v razvojnem okolju Xcode. Izdelava vmesnika ni predstavljala težav zaradi dobre implementacije razvoja metode povleci in spusti (angleško drag & drop). Videz aplikacije je bil najhitreje izdelan v okolju Xcode. Zaradi novega programskega jezika Swift je razvoj logike aplikacije upočasnil izdelavo aplikacije. Slabost domorodne aplikacije je potreba po fizični mobilni napravi zaradi uporabe kamere (ni možna na emulatorju).

V prvih fazah projekta je največ težav povzročila izdelava grafičnega vmesnika. Ogrodje Xamarin.Forms v primerjavi z ostalima okoljema ne podpira grafičnega urejevalnika videza.

Videz smo razvili v datoteki XAML. Ker smo imeli največ znanja s programskem jeziku C#, je razvoj logike potekal najhitreje. Implementacija »custom renderejev« za osnovne lastnosti komponent je bila moteč dejavnik in odvečno delo. Xamarin studio prav tako omogoča celoten razvoj domorodnih aplikacij Android, iOS in Windows Phone. V domorodnih aplikacijah obstaja grafični urejevalnik videza aplikacije.

Aplikacije Xamarin.Forms so med sabo bolj podobne kot domorodne aplikacije, razvite v Android studiu in Xcode. Aplikaciji Xamarin.Forms smo razvili enako hitro kot ostali dve aplikaciji. Pri razvoju projekta Xamarin.Forms smo obe aplikaciji dobili znotraj enega projekta. Čas razvoja je bil v primerjavi z razvojem domorodnih aplikacij Android in platforme iOS enkrat krajši. V primeru naše testne aplikacije se je bolje obnesel večplatformski razvoj (Xamarin.Forms).

Menim, da so aplikacije Xamarin.Forms bolj primerne za poslovne aplikacije. Te namreč uporabljajo pretežno forme za vnos določenih podatkov. Za razvoj »pixel perfect« aplikacij, kot so igre in socialne aplikacije, bi bile razlike na drugih platformah bolj opazne. Hkrati bi uporabniku dale občutek nekonsistentnega videza. Dodatna prednost okolja Xamarin studio je enotni programski jezik C#. Razvijalec se tako lahko bolj specializira za razvoj mobilnih aplikacij. Okolji Xcode in Android studio sta brezplačni. Microsoftov produkt Xamarin studio ima za razvijalce relativno drage licence. Obstaja comunity različica, ki pa za resen razvoj aplikacije ne podpira testiranja, verzioniranja in podobnih ključnih lastnosti za timsko delo.

Literatura

[1] Co., Svetlin Nakov (2013). FUNDAMENTALS OF COMPUTER PROGRAMMING WITH C#. Introduction to Programming (str. 75-79). Sofia, Bulgaria.

[2] Apple CoreData. (december 2016). [Online]. Dosegljivo: https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/CoreData/index.html?utm_source=iosstash.io.

[3] Apple documentation for Swift. (december 2016). [Online]. Dosegljivo: https://developer.apple.com/library/content/documentation/Swift/Conceptual/Swift_Programming_Language/.

[4] Cocoa-and-CocoaTouch. (december 2016). Dosegljivo: http://stackoverflow.com/questions/2297841/cocoa-versus-cocoa-touch-what-is-the-difference.

[5] Developer Android sdk tools. (december 2016). [Online]. Dosegljivo: https://developer.android.com/studio/releases/sdk-tools.html.

[6] Developer Android studio. (december 2016). [Online]. Dosegljivo: https://developer.android.com/studio/intro/index.html.

[7] Developer Android studio features. (december 2016). [Online]. Dosegljivo: https://developer.android.com/studio/features.html

[8] Developer Apple Xcode. (december 2016). [Online]. Dosegljivo: https://developer.apple.com/xcode/ide/.

[9] Developer Apple Xcode features. (december 2016). [Online]. Dosegljivo: https://developer.apple.com/xcode/features/. [Poskus dostopa 14 12 2016].

[10] Developer Xamarin studio. (december 2016). [Online]. Dosegljivo: https://developer.xamarin.com/guides/cross-platform/xamarin-studio/ide-tour/.

[11] Gradle. (december 2016). [Online]. Dosegljivo: http://stackoverflow.com/questions/16754643/what-is-gradle-in-android-studio.

[12] Integrated-development-enviroment. (december 2016). [Online]. Dosegljivo: http://searchsoftwarequality.techtarget.com/definition/integrated-development-environment.

[13] Java native interface. (december 2016). [Online]. Dosegljivo: https://www3.ntu.edu.sg/home/ehchua/programming/java/JavaNativeInterface.html.

[14] Java overview. (december 2016). [Online]. Dosegljivo: https://www.tutorialspoint.com/java/java_overview.htm.

[15] Jetbrains. (december 2016). [Online]. Dosegljivo: https://www.jetbrains.com/idea/.

[16] Media plugin usage. (december 2016). [Online]. Dosegljivo: https://components.xamarin.com/view/mediaplugin.

[17] Mobile-emulator. (december 2016). [Online]. Dosegljivo: https://www.techopedia.com/definition/30676/mobile-emulator.

[18] Microsoft XAML. (december 2016). [Online]. Dosegljivo: https://msdn.microsoft.com/en-us/library/cc295302.aspx.

[19] MVC. (december 2016). [Online]. Dosegljivo: https://www.tutorialspoint.com/struts_2/basic_mvc_architecture.htm.

[20] SQLite. (december 2016). [Online]. Dosegljivo: https://sqlite.org/about.html.

[21] Swift-Toast library. (december 2016). [Online]. Dosegljivo: https://github.com/scalessec/Toast-Swift. [Poskus dostopa 29 12 2016].

[22] Programming language. (december 2016). Dosegljivo: http://www.businessdictionary.com/definition/programming-language.html.

[23] Xamarin platform features. (december 2016). Dosegljivo: https://www.xamarin.com/platform.

[24] XML. (december 2016). Dosegljivo: http://searchsoa.techtarget.com/definition/XML. [Poskus dostopa 10 12 2016].